

目次

インタラクティブオブティマイザの始め方 9

サーフェスリスト 12

サーフェスリスト機能 13

サーフェスの編集 16

Raytrace Mode (光線追跡モード): 17

オブジェクトビューでのみ動作する機能 19

ビットマップモード: 20

サーフェス 23

サーフェス 24

平面サーフェス 24

タイプ A サーフェス: 自由 B- スプラインサーフェス 25

タイプ A サーフェス: X 軸対称および Y 軸対称 B- スプラインサーフェス 26

タイプ A サーフェス: X 軸対称および Y 軸対称 B- スプラインサーフェス 27

タイプ A サーフェス: 双円錐面 28

Type A サーフェス: XY 多項式サーフェス 29

Type A サーフェス: 自由曲面 - 放射タイプ 31

Type A サーフェス: 自由曲面 - スライスタイプ 34

タイプ B サーフェス: 非対称プロファイル 35

タイプ B サーフェス: 対称プロファイル 36

タイプ B サーフェス: 楕円プロファイル 37

タイプ B サーフェス: 自由形状ループ 38

タイプ B サーフェス: 全反射レンズ形状 39

タイプ C サーフェス: 2D パス 40

Type C サーフェス: 3D パス 41

Type C サーフェス: 螺旋パス 42

C タイプサーフェス: 自由螺旋パス 43

C タイプサーフェス: 矩形螺旋パス 44

タイプ C サーフェス: 自由正方スパイラルパス 45

D タイプサーフェス: スプライン反射鏡 46

D タイプサーフェス: コーンリフレクタ 47

セグメント 48

セグメント - ライン	49
セグメント - スプライン	50
セグメント - コーニック	51
セグメント - カスタムフレネル	57
フレネル化	58
非球面にフィット	60

プロパティエディタ 61

プロパティエディタの項目	62
Pick-up ピックアップ	63
関数	65
スライダーホイール	68

オブジェクトビュー 69

編集機能	70
ツールバーコマンド	72

オブジェクトのタイプ 73

オブジェクトタイプ：レンズ	74
オブジェクトタイプ：押し出しオブジェクト	76
オブジェクトタイプ：放射対称オブジェクト	77
オブジェクトタイプ：スイープオブジェクト	79
オブジェクトタイプ：バイアキシャルオブジェクト	80
オブジェクトタイプ：反射板	82
オブジェクトタイプ：TIR レンズ	84
オブジェクトタイプ：自由曲面レンズ	86
オブジェクトタイプ：アドバンスドスイープオブジェクト	87
アドバンスドスイープオブジェクトエディタ：	88
オブジェクトタイプ：レプタイルオブジェクト	95

ソースの定義 100

オブティマイゼーション 102

最適化ウィンドウ	102	Irradiance 放射照度	124
Optimization settings: 最適化の設定	103	Irradiance profile 放射照度プロファイル	125
Downhill Simplex algorithm (ダウンヒルシンプレックスアルゴリズム) の設定パネル	104	Intensity 光度	127
Variable list (変数リスト)	106	Candela profile カンデラプロファイル	128
User-defined variable (ユーザー定義変数): 106		Uniformity 均一度	129
Variables (変数)	107	ビーム幅	131
Operand (オペランド)	108	Beam center ビームセンタ	132
Object list(オブジェクトリスト)	110	Spot size スポットサイズ	133
最適化アルゴリズム	111	Luminance 輝度	134
シンプレックス法 (下り坂法): 111		User defined ユーザー定義	137
変数スキャン: 111		Mathmatic functions 数学関数	138
公差解析 111		Scheme スキーム	139
ソース構成	112	Photorealistic フォトリアリスティック	140
Optimization log (最適化ログ)	113	After-Scheme (アフタースキーム) プログラミング	141
最適化ログ - ユーザーインターフェース	114	3D オプティマイザ関数	142
最適化ログ - モデルの更新	116	cplx 142	
最適化ログ - ログファイル	117	cply 142	
Optimization operand(最適化オペランド) 118		cplz 142	
Flux フラックス	121	cpx 143	
CIE u'v' および CIE xy	122	cpy 143	
		cpz 143	
		edge 144	
		face:asymasphere 144	
		face:biconic 145	

face:freeform	145	surf_prop	158
face:planar	146	surface:set_parameter	158
fpx	146	trajectory	159
fpv	146	trajectory:get_coordinate	159
get_value	146	trajectory:get_length	160
get_controlpoint_position	147	trajectory:get_normal	160
get_ite	147	trajectory:get_position	160
get_object_parameter	148	trajectory:get_tangent	161
get_savepath	148	var	161
get_segment_parameter	148	その他項目	162
get_surface_parameter	149	append	162
get_tp_interface	149	apply_bulk	162
math:spline_integration_split	149	apply_material	162
obj_prop	150	apply_property	163
object:get_parameter	150	applybulk	163
object:set_parameter	150	applymaterial	163
path:get_intersect_plane	151	applyproperty	164
path:get_length	151	getitem	164
path:get_position	152	insert	165
path:get_tangent	152	name	165
property:object	152	pause	166
property:segment	153	prop:add_bulk	166
property:surface	153	prop:add_bulk_gegenbauer	167
seg_prop	154	prop:add_lambertian_scatter_surface	167
segment:set_parameter	154	prop:add_material	168
set_controlpoint_position	154	raytraceflag	168
set_object_parameter	155	removeat	168
set_segment_parameter	155	source:grid_beam_orientation	169
set_surface_parameter	156	var_exists	170
splx	156	wait	170
sply	156	ファイル I/O	171
splz	157	close_all_files	171
spx	157	close_all_files()	171
spy	157	close_file	171
spz	158		

delete_file 171
end_of_file 172
file_exists 172
get_application_folder 172
get_file_extension 173
get_file_name 173
get_file_path 173
get_files 174
get_up_folder 174
make_folder 174
read_file 175
wait_until_file_exists 175
write_file(fid, ln) 176
write_text 176

ファンダメンタル 177

{...} 177
= 177
break 177
比較演算子 178
continue 178
for...next 179
if...else...endif 179
論理演算子 180
while...end while 180

数学演算子 182

abs 182
acos 182
acosd 182
asin 183
asind 183
atan 183
atand 184
average 184
avg 184

cint 185
coordinate 185
coordinate:normal 185
coordinate:origin 186
coordinate:up 186
cos 186
cosd 187
cross 187
cstr 187
dot 188
exp 188
get_length 188
get_position 189
get_projected_vector 189
get_tangent 189
iseven 190
isodd 190
ln 190
log 191
math:rotate 191
math:rotate_vector 191
math:spline_integration_split 192
max 192
min 192
normalize 193
pi 193
point 194
pos 194
pos2d 194
position 195
position2d 195
ray 196
ray:direction 196
ray:is_valid 196
ray:position 197

rect	197	save_iso_candela_polar	209
reset_random	197	save_iso_candela_polar_missed	210
rms	198	save_iso_candela_rectangular	210
rnd	198	save_iso_candela_rectangular_missed	210
sd	198	save_model_view	211
sin	199	save_photorealistic_render	211
sind	199	set_model_view_window_size	212
spline	199	set_view	212
sqrt	200		
sum	200		
tan	200		
tand	201		
ubound	201		
vec	201		
vector	202		

文字列関数 203

asc	203
chr	203
instr	203
lcase	204
len	204
mid	204
replace	205
ucase	205

TracePro の解析にかかわる関数 206

get_scheme_value()	206
save_candela_polar_distribution	207
save_candela_polar_distribution_missed	207
save_candela_rectangular_distribution	208
save_candela_rectangular_distribution_missed	208
save_irradiance_map	209

TracePro ファイルに関する関数 213

close_model()	213
closemodel	213
load_model	213
loadmodel	214

TracePro モデリングに関する関数 215

and	215
bend	215
body_transform	215
copy	216
create_edge	216
create_surface	216
createedge	217
createsurface	217
del	218
delete	218
export_surface_as_surface	218
export_surface_as_wirebody	219
face	219
face:centerposition	220
face:get_normal	220
face:load_facet	221
face:ray_intersect	221
face:ray_pick	222

face:uv_bound 223
 facecenterposition 224
 grotate 224
 grotate_adv 225
 intersect 225
 lrotate 225
 make_fly_eye_lens 226
 make_hex_lens_array 227
 make_lens_array 227
 make_pie 228
 makeellipsoid 228
 makeflyeyelens 229
 makehexlensarray 229
 makelens 230
 makelensarray 230
 move 231
 object:export: 231
 or 232
 ray_test: 232
 remove 233
 scale 234
 seet:skin_wires 234
 sheet:skin_wires_path 234
 slinterface:skinning 235
 subtract 236
 surface:domain 236
 surface:eval_normal 237
 surface:eval_pos 237
 surface:eval_uv: 237
 surface:get_coordinate 238
 surface:get_u_dir 238
 surface:get_v_dir 239
 sweep:vector 239
 tp:edge: 240
 tp:wire_body 240

unite 241

トランスフォーム関数

242

trans:transform 242
 trans:transform_norm 242
 trans_compose 242
 trans_identity 243
 trans_inverse 243
 trans_mapping 244
 trans_rotatex 245
 trans_rotatey 245
 trans_rotatez() 245
 trans_scale 246
 trans_translate 246
 transform:compose 247
 transform:identity 247
 transform:inverse 248
 transform:mapping 249
 transform:rotate 249
 transform:rotatex 250
 transform:rotatey 250
 transform:rotatez 251
 transform:scale 251
 transform:transform 252
 transform:transform_norm 252
 transform:translate 253

インタラクティブオブティマイザ

インタラクティブオブティマイザは、二次元スケッチユーティリティであるインタラクティブオブティマイザの後継となるものです。この新しいツールの開発目的は、モデルの作成および最適化時におけるアプリケーションの可能性を広げ、視覚的な表現を向上させることです。インタラクティブオブティマイザでは、インタラクティブオブティマイザで容易に作成できる放射対称なオブジェクトに加えて、BSpline や 双円錐面など、より多くの非対称ジオメトリがサポートされています。

インタラクティブオブティマイザの理念は、TracePro を簡単かつ快適に使用できる体験を提供することです。このユーティリティの支援により、ユーザーは TracePro でさまざまなジオメトリを作成し、さらに指定された目標に基づいてその性能を最適化することができます。

TracePro の詳細情報やテクニカルサポートが必要な場合は、代理店である株式会社ノーツアンドクロス または Lambda Research Corporation までお問い合わせください。

インタラクティブオブティマイザの始め方

インタラクティブオブティマイザの基本要素はサーフェスです。なぜなら、すべてのオブジェクトはサーフェスによって囲まれているからです。インタラクティブオブティマイザは、さまざまな用途に対応するために、主に 3 つのクラスのサーフェスを提供します。

タイプ A:

パラメータ化された形式またはスプライン曲線で記述できるすべてのサーフェス。現在サポートされているタイプは次のとおりです。

- A. 平面
- B. B- スプラインサーフェス
- C. 双円錐
- D. 自由曲面

タイプ A のサーフェスは、レンズ (両面オブジェクト) および押し出しオブジェクトの構築にのみ使用できます。

タイプ B:

タイプ B のサーフェスは、複数のセグメントで構成される閉じたループを使用して作成されます。各セグメントは、いくつかの選択肢の中から異なるタイプとして指定できます。

- A. 非対称プロファイル
- B. 対称プロファイル
- C. 楕円 (円を含む)
- D. 自由曲面ループ
- E. TIR レンズプロファイル

タイプ B のサーフェスは、放射対称オブジェクト (対称プロファイルのみ)、押し出しオブジェクト、およびスイープオブジェクトの作成に使用されます。

タイプ C:

ソリッドオブジェクトを作成するためのスイープに使用されるパスを定義するための、特別な独立したカテゴリ。

- A. 2D パス
- B. 3D パス

タイプ D:

タイプ D は、特定の厚さを持つ単一セグメントのプロファイルを作成するためのもので、リフレクター設計用です。現在、このカテゴリでサポートされているプロファイルは 2 つあります。

- A. 円錐リフレクター
- B. スプラインリフレクター

上記のサーフェスから作成できるオブジェクトは次のとおりです。

1. 放射対称オブジェクト:

放射対称オブジェクトは、対称プロファイル (タイプ B) または楕円ループから作成できます。対称プロファイルを使用してオブジェクトを作成する場合の回転軸は、楕円プロファイルを使用して作成する場

合とは少し異なります。前者の操作は z 軸に沿って行われますが、後者は選択した楕円プロファイルをその垂直軸を中心に回転させることによって行われます。ただし、選択したプロファイルが楕円プロファイルの場合、生成されるオブジェクトは楕円体または皿状になります。

2. レンズ:

レンズには 2 つの側面が必要ですが、レンズのサーフェスとして使用できるのはタイプ A のサーフェスのみです。便宜上、インタラクティブオプティマイザでは、レンズ要素の作成中に選択したサーフェスを反転させることができます。これにより、サーフェスを修正することなく、レンズを凸面から凹面へ、またはその逆に変換できます。

3. 押し出しオブジェクト:

押し出しオブジェクトは、タイプ B などのプロファイルタイプのサーフェスを法線方向に沿って押し出すことで、3 次元オブジェクトを作成します。結果として得られるオブジェクトは「押し出しオブジェクト」と呼ばれます。スイープまたは押し出しは選択したサーフェスの法線方向にのみ適用されるため、タイプ A およびタイプ B のサーフェスを使用して押し出しオブジェクトを作成できます。

4. スイープオブジェクト:

プロファイルサーフェスをパスに沿ってスイープすることで、より複雑なオブジェクトを作成できます。スイープオブジェクトは、スイープ操作に湾曲したパス (複数のセグメントを含む場合もあります) を使用するため、押し出しオブジェクトよりも汎用性があります。

5. 二軸オブジェクト:

二軸オブジェクトを作成するには、2 つの対称プロファイルを選択します。1 つは長軸に沿ったプロファイル、もう 1 つは短軸に沿ったプロファイルです。短軸は長軸に垂直です。2 つのプロファイルは、スイープ操作によってソリッドオブジェクトを形成するために使用されます。

6. リフレクター:

「リフレクター」は、プロファイルタイプ - リフレクター用に設計されています。インタラクティブオプティマイザの操作コンセプトにより、実際のソリッドジオメトリは、より低いレベルのコンポーネント (サーフェスまたはプロファイル) 上に構築する必要があります。設計要件に合わせて、リフレクターオブジェクトはファセット化のための複数の選択肢を提供します。

7. TIR レンズ:

TIR レンズも専用のオブジェクトタイプであり、TIR レンズプロファイルと連携して動作する必要があります。

インタラクティブオプティマイザのユーザーインターフェースは、以下で構成されています。

- サーフェスリスト
- サーフェス
- プロパティエディタ
- オブジェクトビュー
- オブジェクトタイプ
- ソース
- 最適化
- 最適化ログ
- 最適化オペランド
- アフタースキームプログラミング

以下に、インタラクティブオブティマイザの基本的な操作手順が説明されています。

[サーフェス](#)

[サーフェスリスト](#)

[オブジェクトビュー](#)

[オブジェクトのタイプ](#)

[オブティマイゼーション](#)

[Optimization operand\(最適化オペランド\)](#)

インタラクティブオブティマイザのインターフェース

インタラクティブオブティマイザは、動作中に複数のウィンドウを使用して情報を表示します。TraceProにおいて、さまざまな種類の自由曲面形状の生成や光学システムの性能シミュレーションを支援します。さらに、DDE チャネルを通じて TracePro を操作し、最適化を実行します。

インタラクティブオブティマイザのインターフェースを理解するには、以下のセクションを参照してください。

[「サーフェスリスト」サーフェス \(Surfaces\)](#)

[「プロパティエディタ」\(Property editor\)](#)

[「オブジェクトビュー」\(Object view\)](#)

[「オブジェクトのタイプ」\(Object types\)](#)

[「ソース構成」\(Light source\)](#)

[「オブティマイゼーション」\(Optimization\)](#)

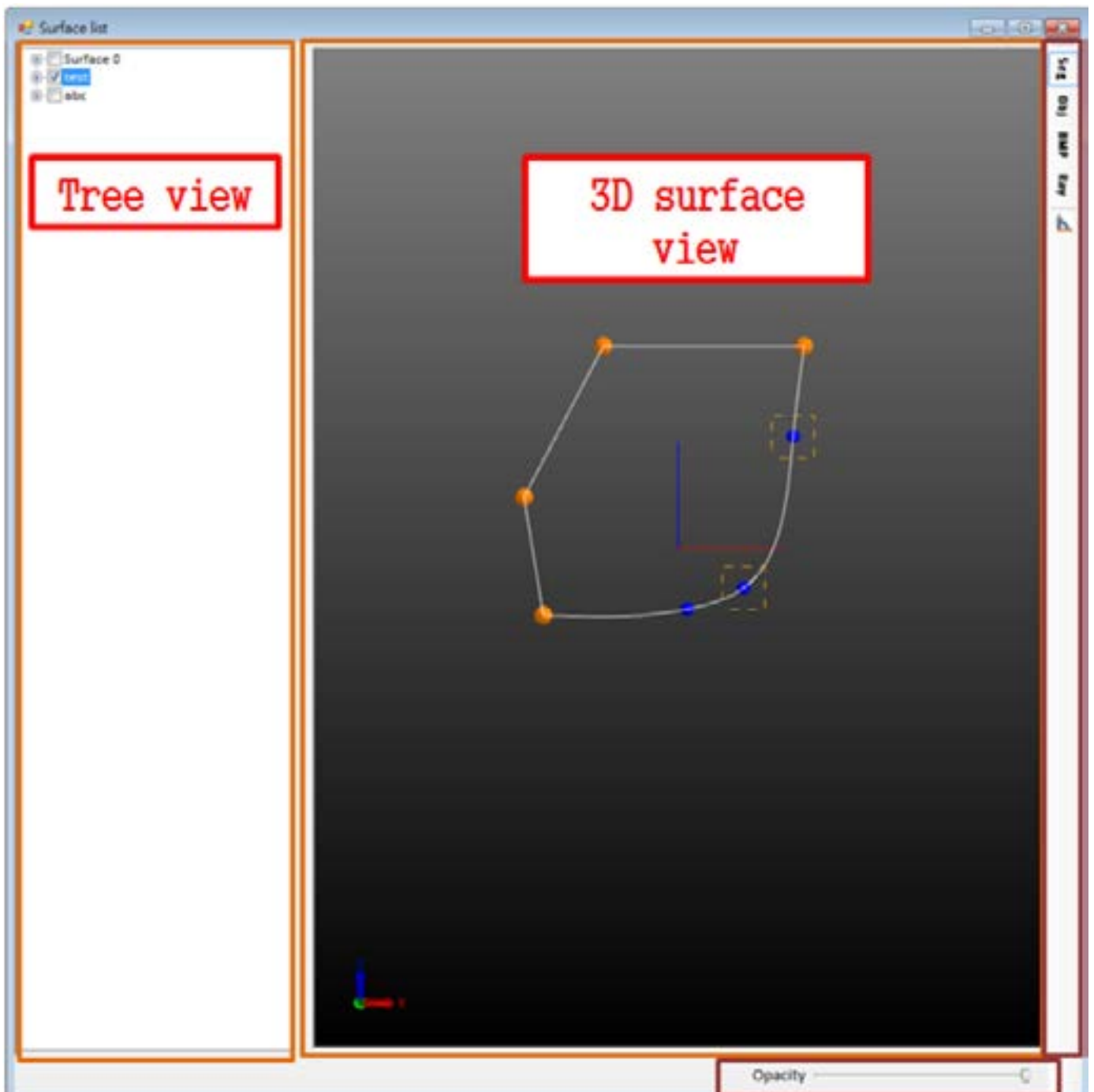
[「Optimization log \(最適化ログ\)」](#)

[「Optimization operand\(最適化オペランド\)」](#)

[「After-Scheme プログラミング」](#)

サーフェスリスト

サーフェスリストウィンドウは、インタラクティブオブティマイザにおいて最も重要で頻繁に使用されるウィンドウです。ツリービューと3次元ビューで構成されています。ツリービューには作成されたすべてのサーフェスがリスト表示され、サーフェスを編集するためのインタラクティブ操作のほとんどは、3D サーフェスビューで行われます。



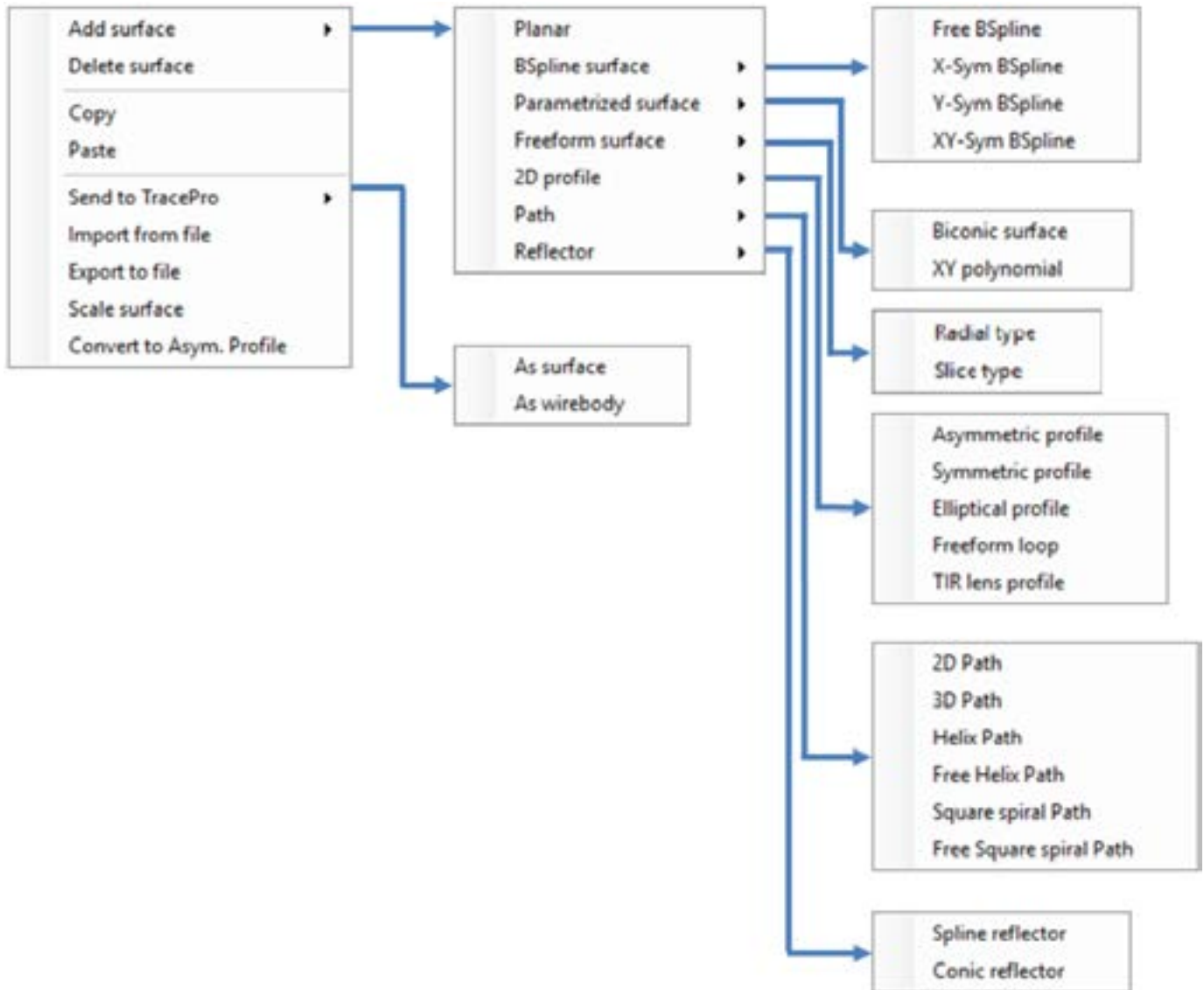
さらに

[Surface List \(サーフェスリスト機能\)](#)

[Additional viewing features \(その他の表示機能\)](#)

サーフェスリスト機能

各サーフェスタイプによって、3D ビューと右端に表示されるメニューパネルは若干異なります。
ツリービュー領域でマウスの右ボタンをクリックすると、ポップアップメニューが開きます。これには、
サーフェスリストウィンドウで使用可能な機能が表示されます。



サーフェスの追加	インタラクティブオブティマイザでサポートされているサーフェスは以下の通りです。 • タイプ A - o 平面 - o 自由 B- スプライン - o X 対称 B- スプライン & Y 対称 B- スプライン - o XY 対称 B- スプライン - o 双円錐サーフェス - o XY 多項式 - o 自由曲面 - 放射状タイプ - o 自由曲面 - スライスタイプ • タイプ B - o 非対称プロファイル - o 対称プロファイル - o 楕円プロファイル - o 自由形状ループ - o TIR レンズプロファイル • タイプ C - o 2D パス - o 3D パス - o ヘリックスパス - o 自由ヘリックスパス - o 矩形スパイラルパス - o 自由矩形スパイラルパス • タイプ D - o スプライン反射器 - o 円錐反射器
サーフェスの選択	インタラクティブオブティマイザでサーフェスを編集する際、タイプ A サーフェスは 1 つ選択できます。タイプ B またはタイプ C サーフェスを編集する場合は、任意の数を選択できます。 ツリービュー内の項目をクリックして、編集するサーフェスを選択します。タイプ B、タイプ C、タイプ D サーフェスを編集する場合、Ctrl キーを押しながらクリックすると、クリックした項目を選択または既存の選択プールから選択解除できます。選択されたサーフェスは、3D サーフェスビューでハイライト表示され、編集可能になり、プロパティエディタは現在選択されているサーフェスに合わせて更新されます。
サーフェスの削除	マウスがサーフェス上にあるとき、ポップアップメニューの「Delete surface? (サーフェスを削除しますか?)」という選択肢が利用可能です。
サーフェスのコピー	Shift キーを押しながらツリービューを左クリックすることで、複数のサーフェスを選択できます。Ctrl-C を使用して、1 つ以上のサーフェスを Windows クリップボードにコピーできます。 サーフェスはクリップボード内でプレーンテキストとして表現されるため、コピーしたサーフェスをインタラクティブオブティマイザまたは任意のテキストエディタに貼り付けることができます。
サーフェスの貼り付け	サーフェスがクリップボードに保存されている場合、ポップアップメニューの「サーフェスを貼り付けますか?」という選択肢が利用可能です。ツリービューにフォーカスがある場合、ホットキー Ctrl-V を使用して貼り付けることもできます。

TracePro への送信	インタラクティブオプティマイザで定義された各サーフェス要素は、TracePro に直接送信できます。これを行うために、それらをソリッドジオメトリにする必要はありません。ただし、TracePro で作成されるエンティティタイプは、インタラクティブオプティマイザのサーフェスタイプによって異なります。 タイプ A およびタイプ B サーフェスは、TracePro に送信されるとボディエンティティになります。2D パスおよび 3D パスの場合は、TracePro にワイヤボディエンティティが作成されます。サーフェス要素は直接 TracePro に送信できるため、TracePro の Scheme 言語を使用して、作成されたサーフェスにさらに処理を加えることができます。
ファイルからインポート	テキストファイルを B- スプラインサーフェスまたはタイプ B プロファイルサーフェスとしてロードできます。ファイルの形式は、<u>B- スプラインサーフェスファイル形式</u>および<u>プロファイルサーフェスファイル形式</u>に記載されています。
ファイルへエクスポート	選択したサーフェスの現在の設定をテキストファイルに保存するには、B- スプラインサーフェスファイルの形式が <u>B- スプラインサーフェスファイル形式</u>に、タイプ B プロファイルサーフェスファイルの形式が<u>プロファイルサーフェスファイル形式</u>に記載されています。
サーフェスの名前変更	サーフェスの名前を編集するには、プロパティエディタを使用する必要があります。サーフェス名のツリービュー表示は編集できません。
表示調整	表示オプションを選択するには、サーフェスリストウィンドウを右クリックしてポップアップメニューを開きます。そこで、「照明」と「サーフェスを表示」を有効にできます。「その他の表示機能」を参照してください。」
サーフェスのスケール	選択したサーフェスを特定の原点と各軸のスケールリングファクタでスケールします。
非対称プロファイルへ変換	選択したプロファイルが対称プロファイルの場合にのみ、この機能が表示されます。選択した対称プロファイルを非対称プロファイルに変換できます。

サーフェスの編集

サーフェスエディタで選択されたサーフェスに対する基本的な操作は、主にマウスで行います。ただし、アクティブなモードに応じて、選択されたサーフェスに対して異なる機能が許可されます。

マウスがコントロールポイントまたはセグメント上にあるとき、オブジェクト上のマウスカーソルがハイライト表示され、クリックするとそのオブジェクトが選択され、その情報がプロパティウィンドウに表示されます。

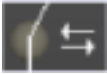
左ボタンを押しながらマウスを動かすと、選択されたコントロールポイントを移動できます。ただし、アクティブなモードがオブジェクトモード（タイプ B およびタイプ C のサーフェスの場合）の場合、コントロールポイントを移動すると、サーフェスオブジェクト全体が同時に移動することに注意してください。

操作モード：

サーフェスの性質が異なるため、選択されたサーフェスのタイプに応じて、有効な操作モードが変化します。有効なすべてのモードは、サーフェスエディタの右側のフレーム境界に表示されます。

次の表は、すべてのボタンとその説明を示しています。

	Segment mode セグメントモード	このモードは、選択されたタイプ B またはタイプ C のサーフェス内のセグメントを修正するために使用します。サーフェスを構成する個々の線分や曲線などを編集する際に有効になります。
	Object mode オブジェクトモード	このモードがアクティブな場合、マウスをドラッグするとオブジェクト全体が移動します。タイプ B およびタイプ C のサーフェスでのみサポートされています。サーフェス全体の位置を調整する際に使用します。
	Bitmap mode ビットマップモード	クリップボードからビットマップ画像をサーフェスエディタの背景として貼り付けるためのモードです。サーフェスの形状を調整する際のガイドとして画像を使用する場合などに便利です。
	Measure 計測	計測ツールを有効にします。サーフェスビュー内の任意の点をクリックし、次に別の点をクリックすると、クリックした 2 点間の距離を測定できます。Shift キーを押しながら操作すると、複数のセグメントを連続して計測でき、同時にセグメント間の角度も測定されます。
	Raytrace mode 光線追跡モード	選択されたタイプ B サーフェスに対して光線を追加したり、選択されたサーフェスのセグメントを変更したりする際に、光線の経路がリアルタイムに変化する様子を確認できるモードです。照明設計や光学系の調整などに役立ちます。
	Clockwise rotate 時計回りに回転	タイプ B サーフェス編集中にこのボタンを押すと、ビューが時計回りに 90 度回転します。
	XZ View XZ ビュー	正の X 軸を上、正の Z 軸を右に表示します。
	YZ View YZ ビュー	正の Y 軸を上、正の Z 軸を右に表示します。
	YX View	Y 軸上向き、X 軸右向きビュー 正の Y 軸を上、正の X 軸を右に表示します。

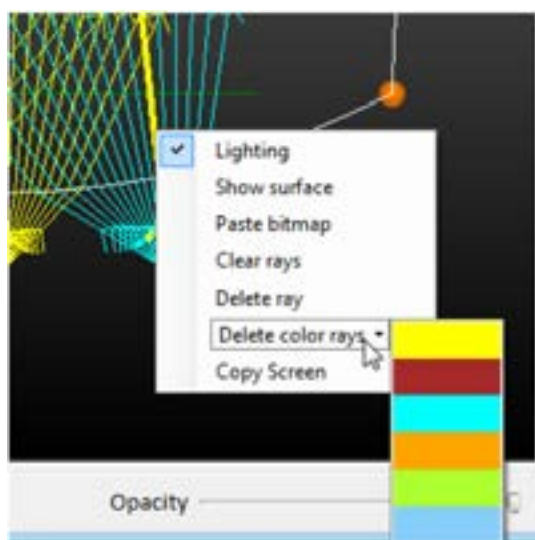
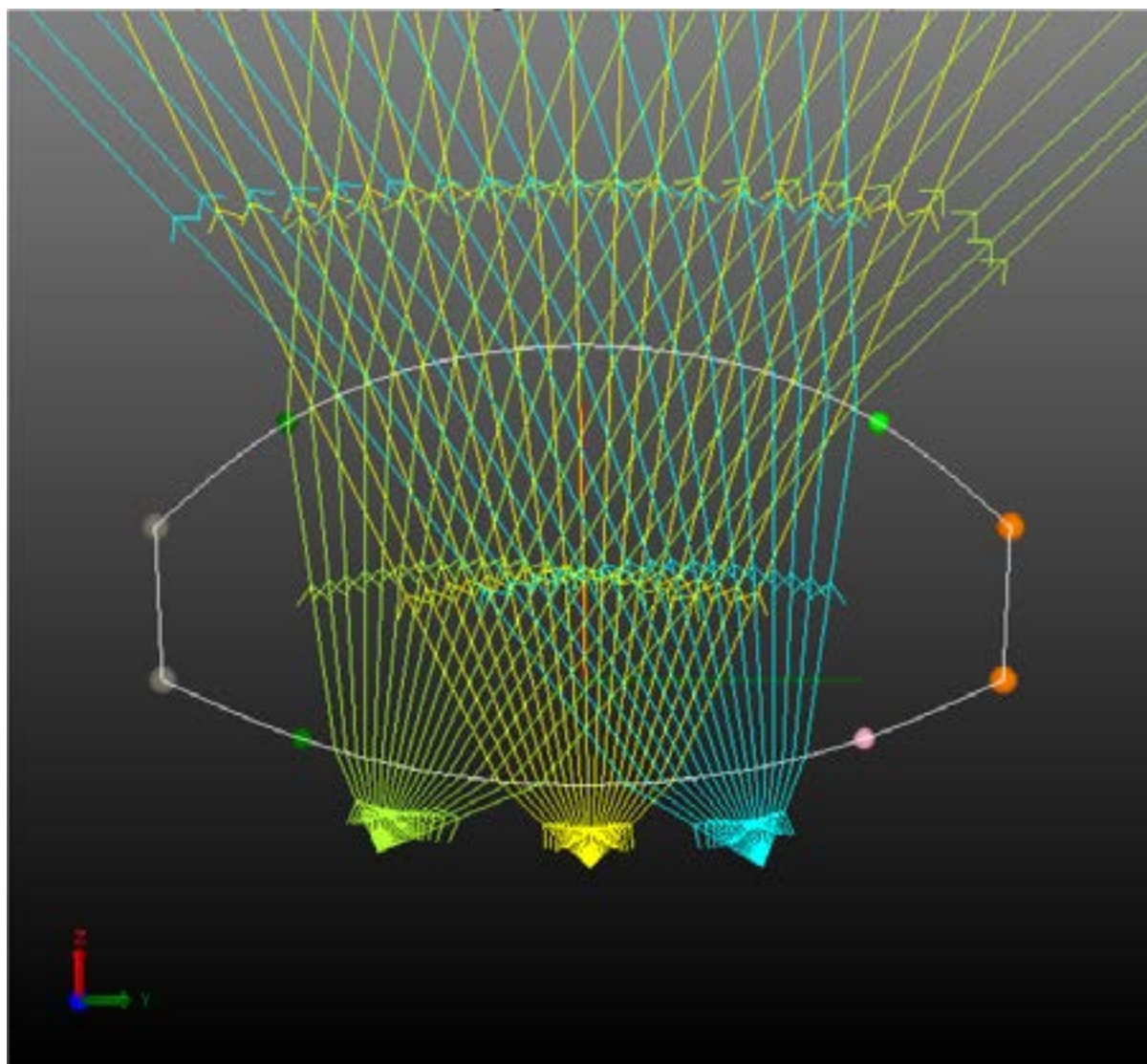
	Isometric View 1 アイソメトリックビュー 1	正の Y 軸を上、正の X 軸を画面奥方向、正の Z 軸を画面手前方向に表示します。
	Isometric View 2 アイソメトリックビュー	正の Y 軸を上、正の X 軸と正の Z 軸を画面奥方向に表示します。
	Profile view プロファイルビュー	このアイコンは、自由曲面（放射状タイプ）のプロファイルを選択している場合にのみ表示されます。選択されたプロファイルの側面図にビューを回転させます。
	Show control point info コントロールポイント情報を表示	このオプションをチェックすると、サーフェスビュー内のコントロールポイントの横に、そのコントロールポイントがミラーリングされているか、および / またはピックアップ定義を持っているかを示す記号が表示されます。 Mirrored（ミラーリングされている場合）： Pickup（ピックアップの対象となっている場合）：

上記の機能に加えて、以下の操作もサーフェスエディタで動作します。

- マウスホイールでズームイン・ズームアウト
- 左マウスボタンでドラッグして視点を回転
- 左マウスボタンをダブルクリックして、視点の中心を原点に移動
- Ctrl キーを押しながら左マウスボタンでドラッグして視点を平行移動
- Ctrl+P で前の制御点を選択
- Ctrl+N で次の制御点を選択

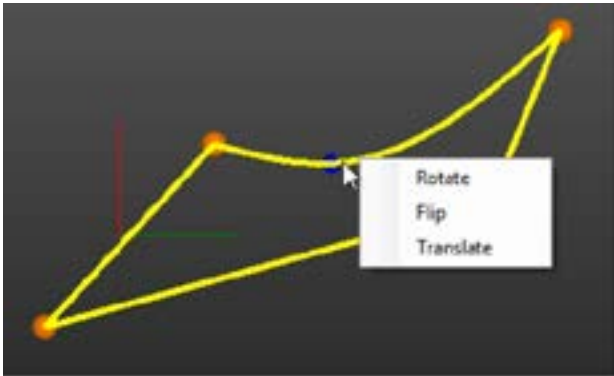
Raytrace Mode（光線追跡モード）：

「光線追跡」モードは、タイプ B サーフェス、つまりプロファイルタイプのサーフェスでのみ動作します。光線がどのように進むかの直感的な図を提供します。光線を追加するためのサポートされている方法には、単一光線、光線ファン、光線シーケンスがあります。選択可能な光線の色は6つあり、同時に、異なるソースからの光線を識別したり、特定の光線を一度に削除したりする必要がある場合に、色によるグループ化機能を提供します。

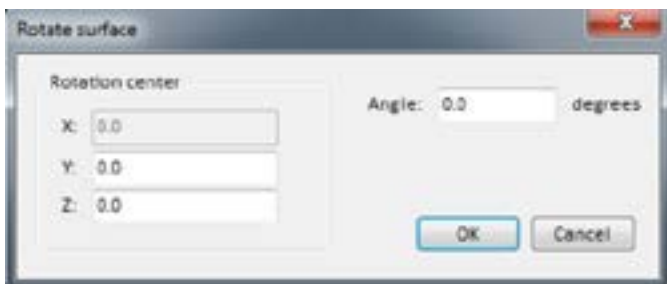


オブジェクトビューでのみ動作する機能

「オブジェクトモード」で、マウスカーソルがサーフェスのプロファイル上にあるとき、サーフェス全体のプロファイルがハイライト表示されます。右クリックすると、選択されたサーフェス全体に対して操作できるいくつかの機能を含むポップアップメニューが表示されます。

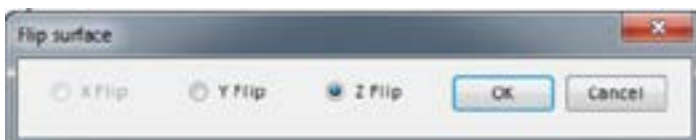


回転

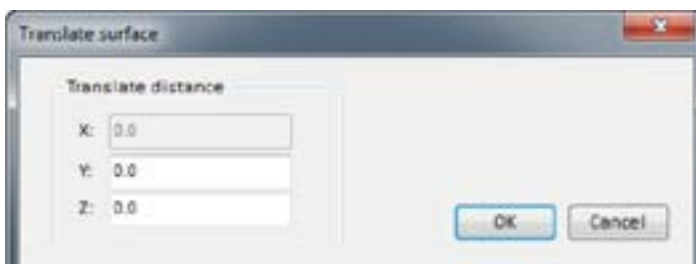


中心は回転の中心です。サーフェスは平面上にあるため、割り当てる必要があるのは2つの成分のみです。回転角の単位は度です。この機能は非対称プロファイルでのみ動作します。

フリップ



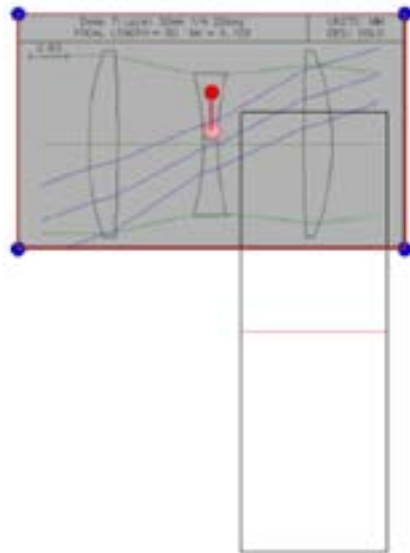
併進移動



Tracelate distance はサーフェスの移動距離です。

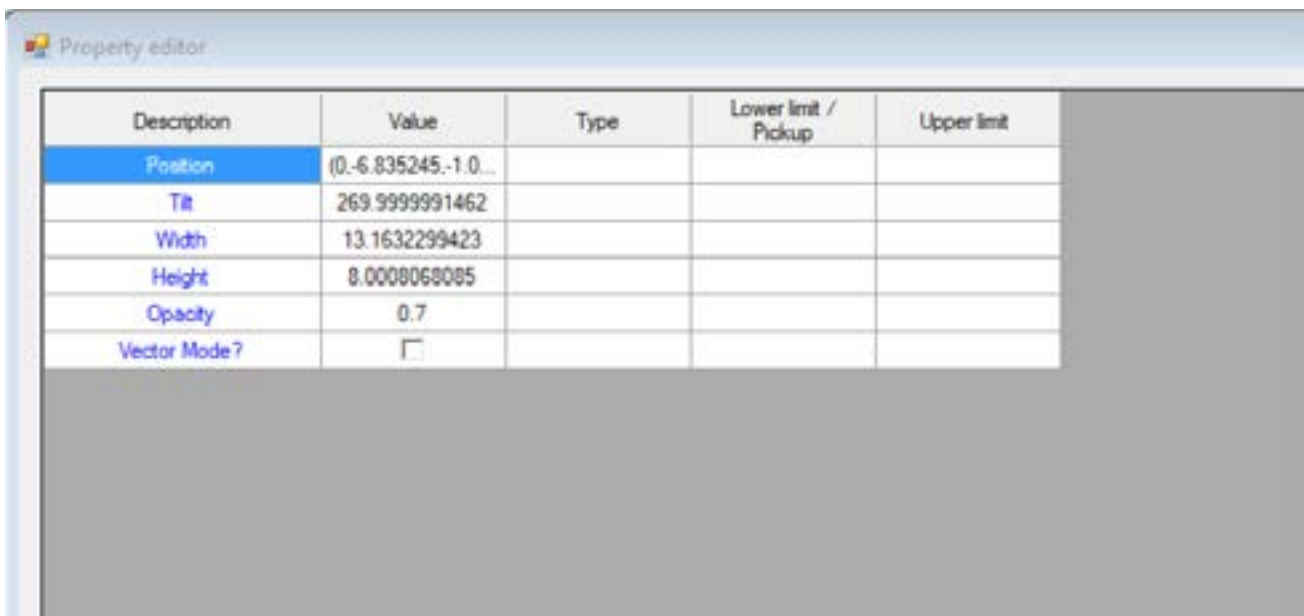
ビットマップモード：

ビットマップは、既存のデザインのプロファイルをスケッチするためのアウトラインを描画する際に、優れたレファレンスとなります。ビットマップモードに切り替える際、Windows のクリップボードにビットマップが検出された場合、ポップアップメニューの項目「ビットマップを貼り付け」がアクティブになります。保存されたビットマップは、現在の作業中のサーフェスビューに貼り付けられます。ビットマップを貼り付けることができるのは、サーフェスが最初に作成され、サーフェスプロファイルサーフェスである場合に限られることに注意してください。



操作可能な 4 つの青い角点は、貼り付けられたビットマップのサイズを変更するためのものです。中央のピンク色の制御点は、ビットマップを移動させるためのものです。そして、赤い方向指示点は、プロットを回転させるためのものです。最も基本的な操作は、これら 6 つの制御点を動かすことによって、貼り付けられたビットマップを実際の座標に合わせて移動、回転、拡大縮小することです。

中央の点を右クリックすると、ポップアップメニューが表示され、ビットマップの削除とコピー機能を利用できます。一方、中央の点をクリックすると、関連するプロパティがプロパティウィンドウに表示されます。



Description	Value	Type	Lower limit / Pickup	Upper limit
Position	(0.6835245,-1.0...			
Tilt	269.9999991462			
Width	13.1632299423			
Height	8.0008068085			
Opacity	0.7			
Vector Mode?	☐			

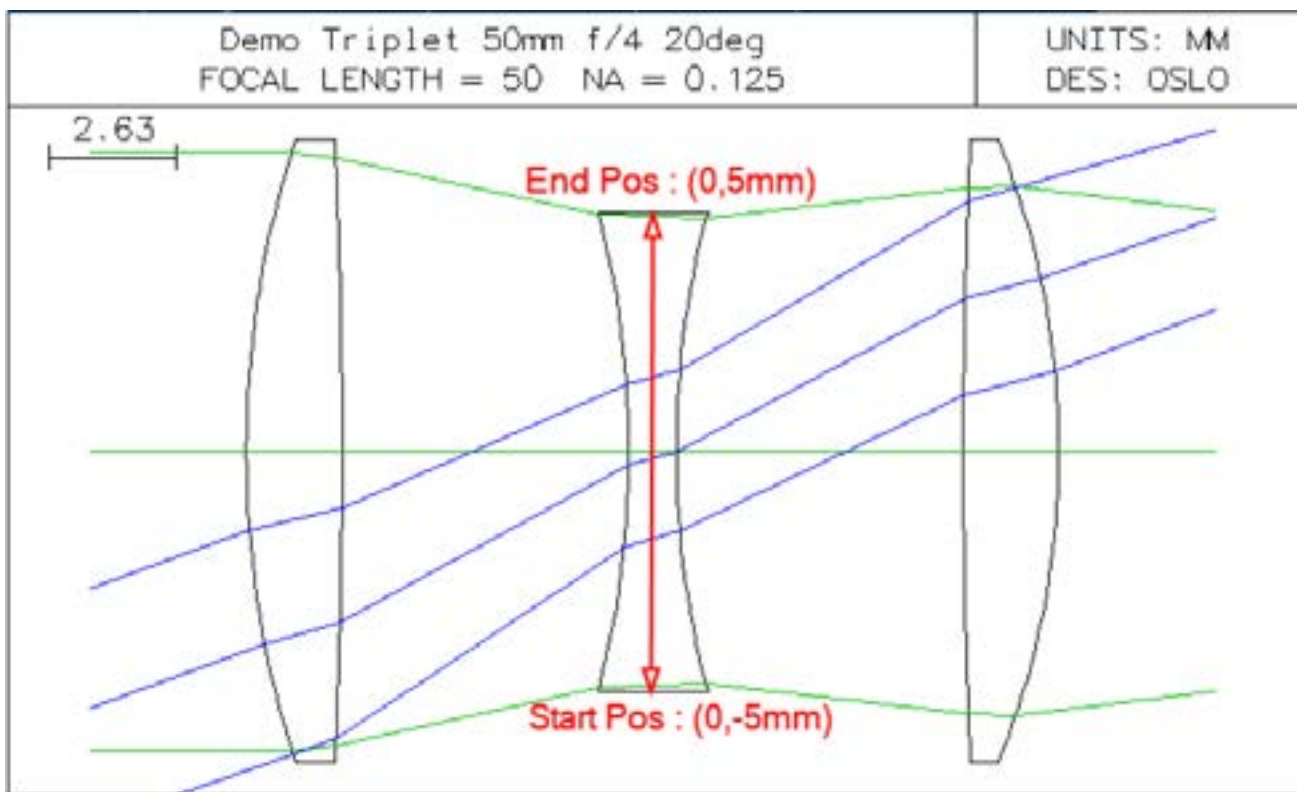
ビットマップを設定する別のモードとして、ベクターモードがあります。「“Vector mode?” ベクターモード？」のオプションがチェックされている場合、プロパティエディタに表示される定義済みの位置にマッピングされているビットマップ上の開始ピクセルと終了ピクセルを示す黄色の矢印がビットマップ上に

表示されます。

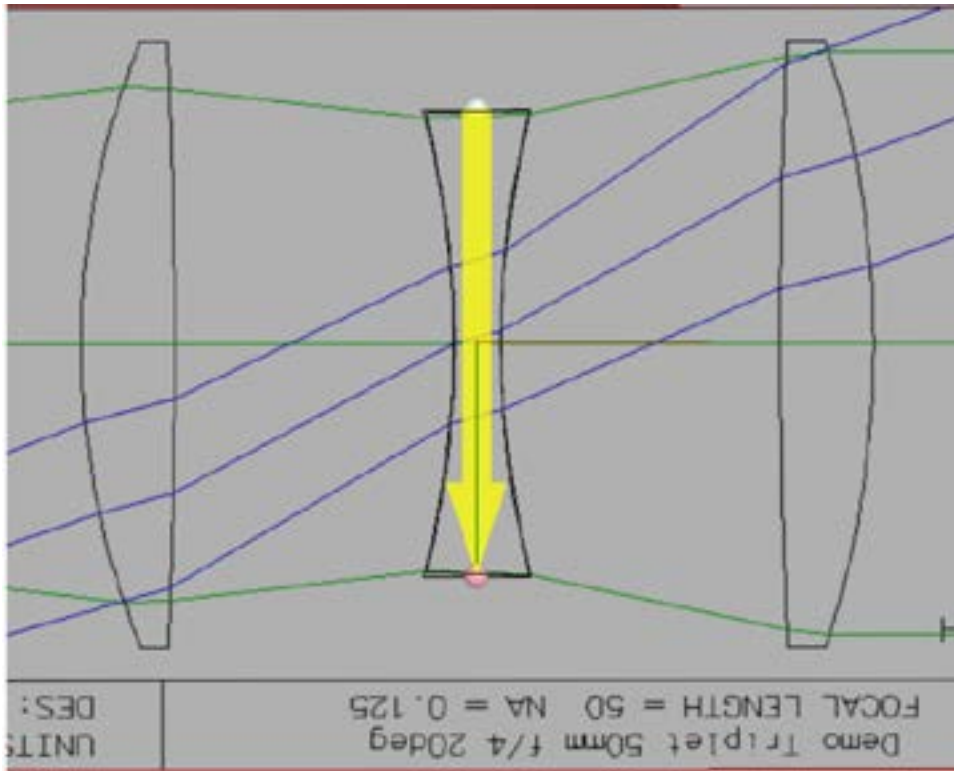
三枚玉レンズを例にとってみましょう。2枚目のレンズの開口半径は5mmなので、それぞれ開始Yを-5.0、開始Zを0.0、終了Yを+5.0、終了Zを0.0と入力します。

Property editor				
Description	Value	Type	Lower limit / Pickup	Upper limit
Start Y	-5			
Start Z	0			
End Y	5			
End Z	0			
opacity	0.33			
Vector Mode?	☒			

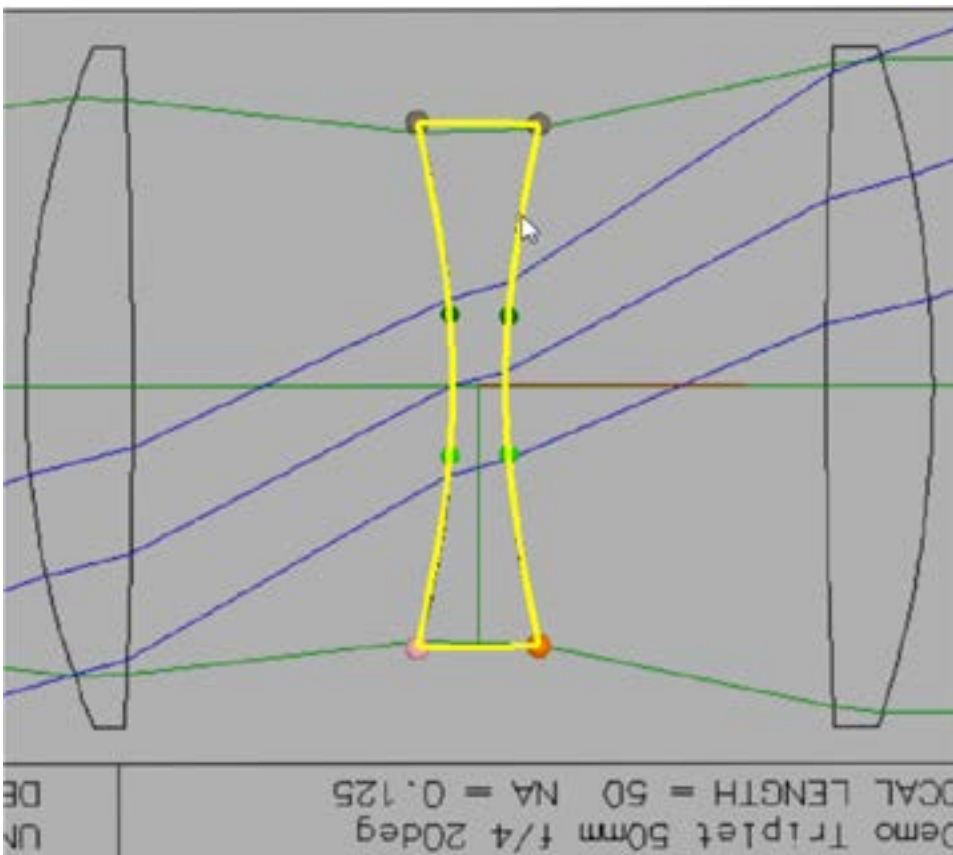
貼り付けられたビットマップ上の対応する開始位置と終了位置は、以下の通りです。



それから、開始点（黄色の矢印の尾部）をビットマップ上の開始位置までドラッグし、終了点（黄色の矢印の頭部）をビットマップ上の終了位置までドラッグします。



これでビットマップはスケッチウィンドウの正しい位置に、正しいスケールで適切に配置されます。セグメントモードに切り替え、その後、セグメントを修正して、貼り付けられたプロファイルに完全に一致させます。



現在の TracePro モデルビューのスクリーンショットを貼り付けるには、ポップアップメニューの「**Paste TracePro screenshot**」(TracePro スクリーンショットを貼り付け)をクリックしてください。モデルビューのスクリーンショットを貼り付けた後、ビットマップの原点はプロファイル編集座標の原点に配置され、ビットマップのサイズは編集座標に合うように調整されます。この機能を使用する前に、TracePro モデルビューの表示方向を次のように変更することをお勧めします 。

サーフェス

サポートされているサーフェスは、メジャーな3つのクラスに分類されます。

- タイプ A
 - 平面
 - フリー B スプライン
 - X 対称 B スプライン & Y 対称 B スプライン
 - XY 対称 B スプライン
 - 双円錐面
 - XY 多項式
 - フリーフォームサーフェス - 放射型
 - フリーフォームサーフェス - スライス型
- タイプ B
 - 非対称プロファイル
 - 対称プロファイル
 - 楕円プロファイル
 - フリーフォームループ
 - TIR レンズプロファイル
- タイプ C
 - 2D パス
 - 3D パス
 - ヘリックスパス
 - フリーヘリックスパス
 - スクエアスパイラルパス
 - フリースクエアスパイラルパス
- タイプ D
 - スプライン反射鏡
 - 円錐反射鏡

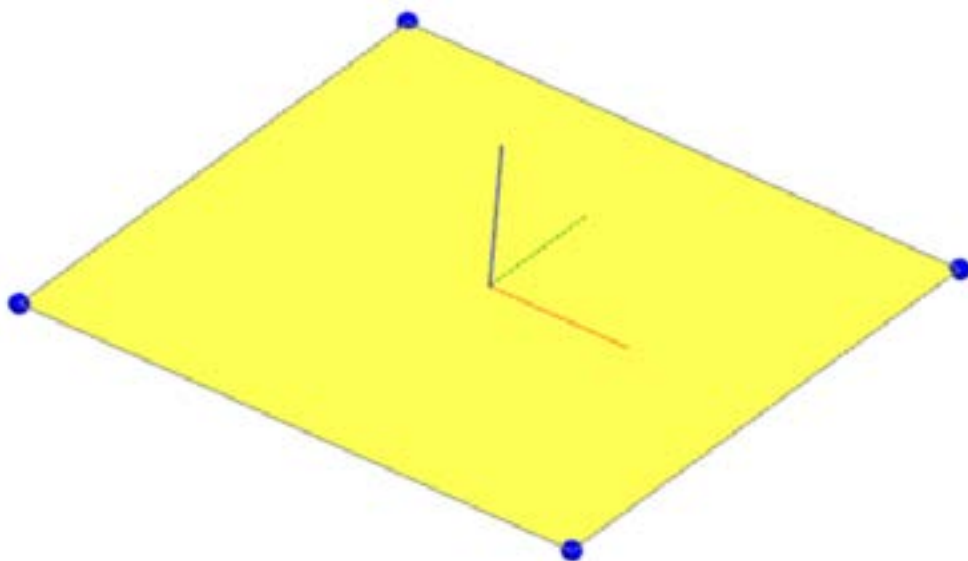
詳細：

- 平面
- フリー B スプライン
- X 対称 B スプライン & Y 対称 B スプライン
- XY 対称 B スプライン
- 双円錐面
- XY 多項式
- フリーフォームサーフェス - 放射型
- フリーフォームサーフェス - スライス型
- 非対称プロファイル
- 対称プロファイル
- 楕円プロファイル
- フリーフォームループ
- TIR レンズプロファイル
- 2D パス
- 3D パス
- ヘリックスパス
- フリーヘリックスパス
- スクエアスパイラルパス
- フリースクエアスパイラルパス
- スプライン反射鏡
- 円錐反射鏡
- 「Type A サーフェス：平面」

サーフェス

平面サーフェス

平面サーフェスは最も基本的な Type-A サーフェスであり、四隅の 4 つの制御点を移動させることはできません。平面サーフェスの調整可能なプロパティは、サーフェスサイズとサーフェスプロパティのみです。



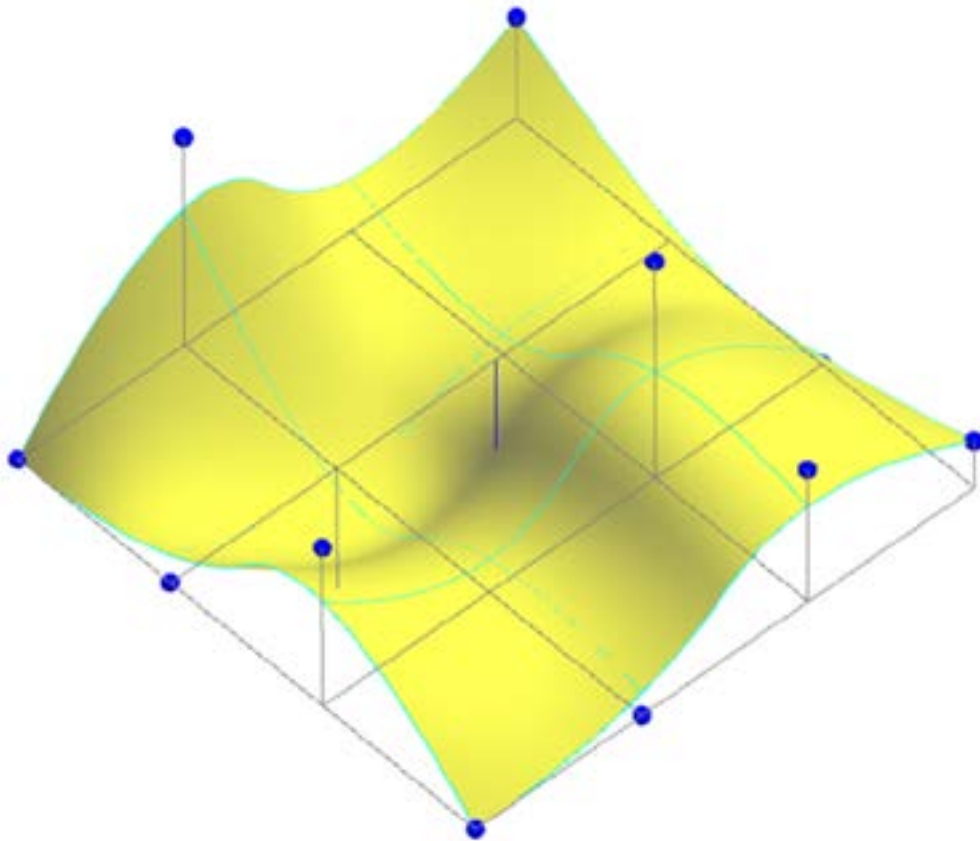
タイプ A サーフェス：自由 B- スプラインサーフェス

B- スプラインサーフェスは、グリッドの交点に制御点を持ち、行と列の数はいつでも変更できます。ただし、グリッドノードの数を変更すると、サーフェスタイプは平面にリセットされます。

B- スプラインサーフェスのもう一つの重要な係数は、ノット次数です。ノット次数は、ローカルな X 方向と Y 方向で独立して指定できます。

便宜上、新しいサーフェスを追加する際に B- スプラインサーフェスタイプを指定できます。

すべての制御点を個別に調整できます。B- スプラインは、連続的で滑らかな曲面を作成するための最も汎用性の高いサーフェスタイプです。ただし、B- スプラインサーフェスタイプは、最適化中に最も制御が難しいものでもあります。



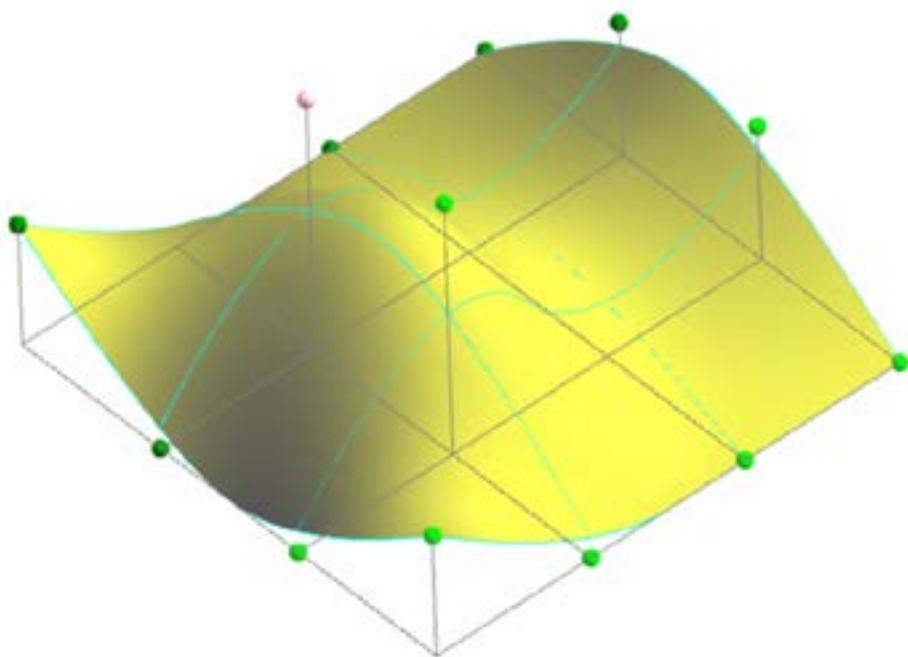
タイプ A サーフェス：X 軸対称および Y 軸対称 B- スプラインサーフェス

B- スプラインサーフェスは、格子交点に制御点を持ち、行と列の数はいつでも変更できます。ただし、格子ノード数を変更すると、サーフェスタイプは平面にリセットされます。

B- スプラインサーフェスのもう一つの重要な係数は、ノット次数です。ノット次数は、ローカルな X 方向と Y 方向で個別に指定できます。

便宜上、新しいサーフェスを追加する際に B- スプラインサーフェスのタイプを指定できます。

軸に沿って対称性を持つサーフェスを作成したい場合は、あらかじめ定義された対称性を持つサーフェスタイプを使用してください。B- スプラインサーフェスが X 軸対称、つまり $z(x,y) = z(-x,y)$ である場合は、X 軸対称 B- スプラインサーフェスタイプを使用します。一方、制御点が $z(x,y) = z(x,-y)$ の関係を満たす場合は、Y 軸対称 B- スプラインサーフェスタイプを使用します。

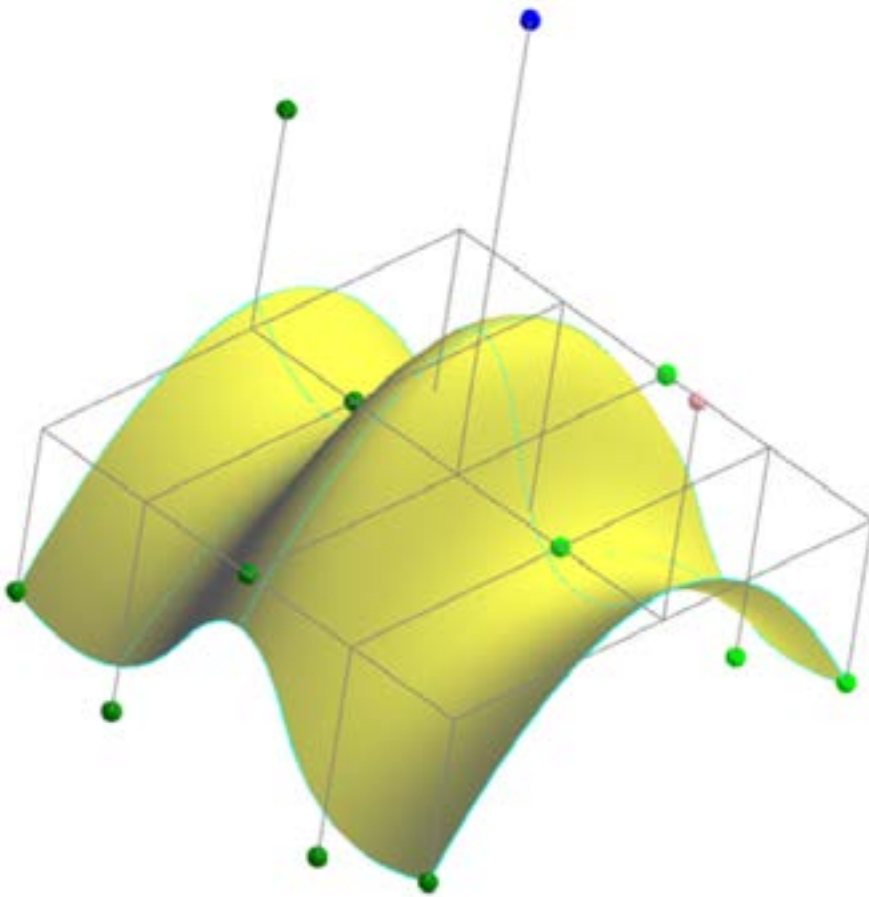


タイプ A サーフェス：X 軸対称および Y 軸対称 B- スプラインサーフェス

B- スプラインサーフェスは、格子点の交点に制御点を持ち、行と列の数はいつでも変更できます。ただし、格子点の数を変更すると、サーフェスタ입は平面にリセットされます。

B- スプラインサーフェスのもう一つの重要な係数は、ノット次数です。ノット次数は、ローカルな X 方向と Y 方向に対して個別に指定できます。

X 軸と Y 軸の両方に対称性が必要な場合は、XY-Sym B- スプラインサーフェスタ입を使用してください。



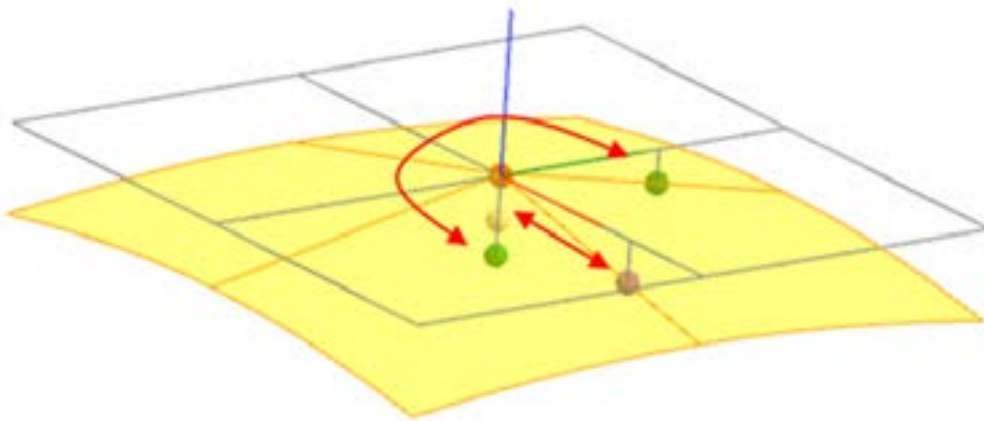
タイプ A サーフェス：双円錐面

B 双円錐面は、非対称照明の用途で一般的に使用されます。対称なソースを使用した場合、異なる曲率と円錐定数によって光線が非対称に曲げられ、非対称な照明パターンが生成されます。

双円錐面は、そのパラメータ化された性質により、B- スプライン面とは異なります。その形状は、2つの軸に沿った曲率と円錐定数によって決定されます。3つの独立した制御点を調整することで、円錐形状のパラメータセットを決定できます。したがって、双円錐面は、X 方向と Y 方向の 2つのパラメータセットを決定するために、合計 5つの制御点を必要とします。

双円錐面は、X 子午線と Y 子午線の両方に円錐項を持ちます。

$$z = \frac{CV_x x^2 + CV_y y^2}{1 + \sqrt{1 - CV_x^2 (CC_x + 1)x^2 - CV_y^2 (CC_y + 1)y^2}}$$

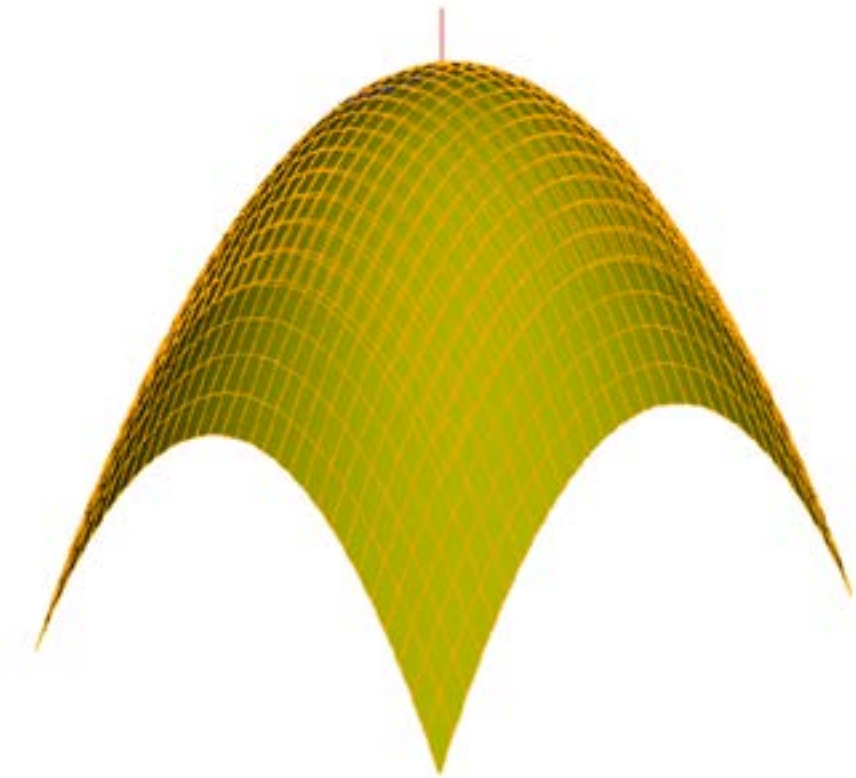


Type A サーフエス：XY 多項式サーフェス

自由曲面は、非対称光学系の設計によく用いられます。そのサグ方程式は、x-y の多項式で表されます。

$$z = c_{00}x^0y^0 + c_{10}x^1y^0 + c_{01}x^0y^1 + c_{11}x^1y^1 + c_{20}x^2y^0 + \dots + c_{mn}x^m y^n + \dots$$

自由曲面は、Interactive Optimizer における他のサーフェスタイプとは異なります。なぜなら、サーフェス形状を調整するための制御点を持たないからです。



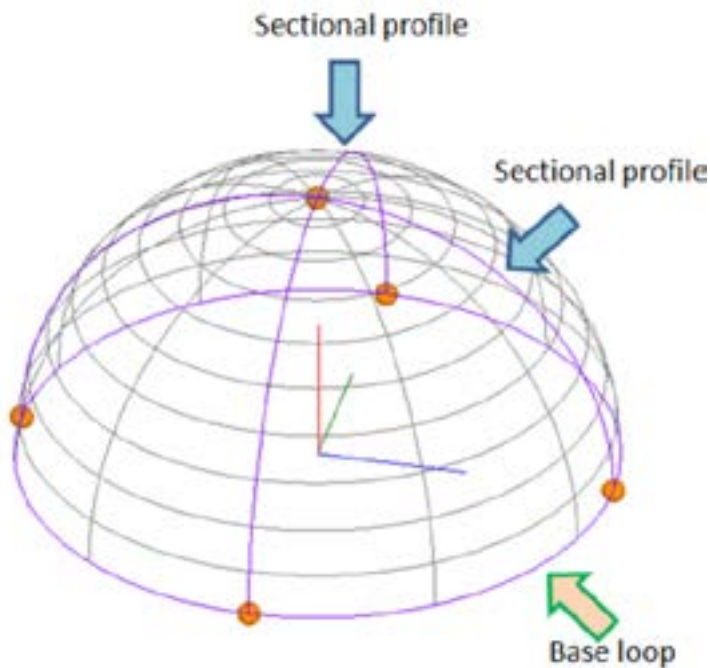
サーフェス形状を設定する唯一の方法は、サーフェス方程式のパラメータを入力することです。また、「Setup button (セットアップボタン)」を使用して、サーフェスを構成するために使用したい項の数を定義することもできます。

Description	Value	Type	Lower limit / Pickup	Upper limit
Name	Surface 0			
Surface type	XY polynomial			
Surface ID	0			
Surface catalog				
Surface property				
Reflective?	☐			
Size	(20,20)			
Define terms	Setup			
X ⁰ *Y ⁰	0.0	Specified		
X ¹ *Y ⁰	0.0	Specified		
X ⁰ *Y ¹	0.0	Specified		
X ² *Y ⁰	0.0	Specified		
X ¹ *Y ¹	0.0	Specified		
X ⁰ *Y ²	0.0	Specified		

The 'Solve' dialog box is shown with the 'Solve' button highlighted in red. The 'Variables' section contains a grid of checkboxes for variables X^0 through X^5 and Y^0 through Y^5 . The checkboxes for Y^0 , Y^1 , and Y^2 in the X^0 column are checked. The 'Apply' and 'Cancel' buttons are at the bottom.

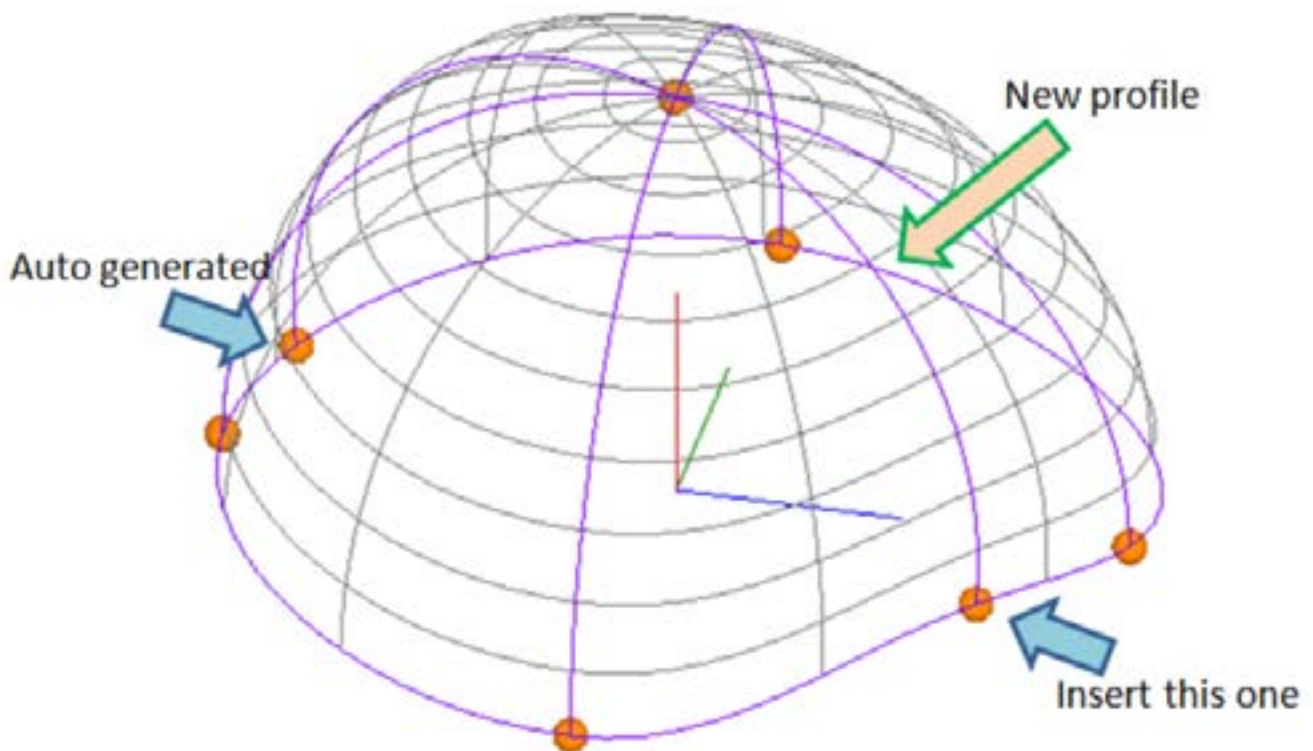
Type A サーフェス：自由曲面 - 放射タイプ

放射タイプの自由曲面は、一つのベースループと複数の断面プロファイル（2つ以上）によって構築されます。



基本ループは平面上にあり、すべての断面プロファイルが結合する頂点は原点を通る垂直線上にあります。これにより、自由曲面 - 放射状タイプは半球状の空間のみを占めることができます。

基本ループにコントロールポイントを追加することは、新しい断面プロファイルを作成することです。どのプロファイルも基本ループに結合する必要があるため、基本ループにコントロールポイントを追加すると、反対側にも対応するコントロールポイントが自動的に挿入されます。



それぞれの断面プロファイルは、一連の制御点によって制御される連続した曲線です。結果として得られる全サーフェスは、すべてのプロファイルの組み合わせです。

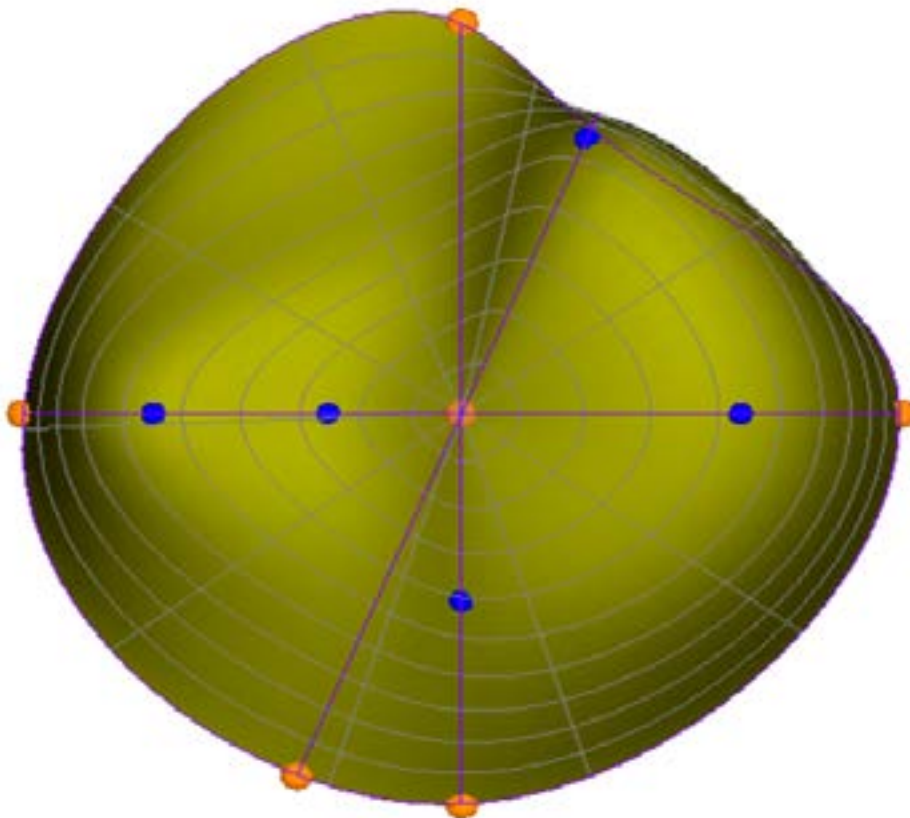
3D 空間におけるコントロールポイントの移動の困難さに関して、プロファイルビューは、特定のプロファ

イルを編集するためにユーザーが適切な視点を設定するためのショートカットとなります。

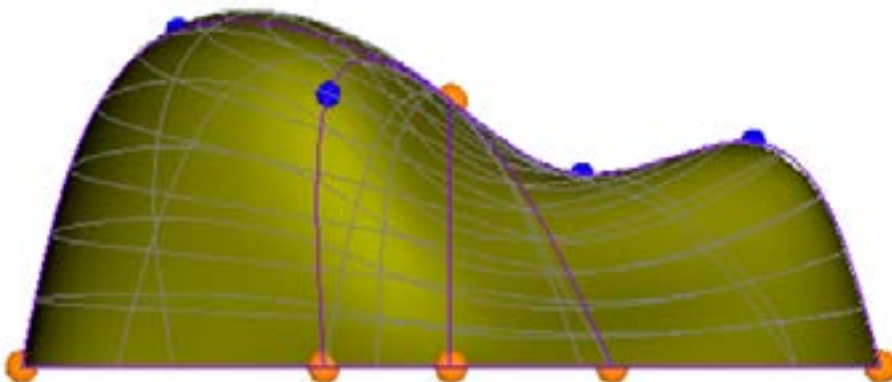
ベースループを含む任意のプロファイルをマウスでクリックすると、サーフェスリストウィンドウの右端にある垂直メニューバーに追加のボタンが表示されます。この**プロファイルビュー**ボタンは、選択されたプロファイルがビュー平面に配置されるように、ビューを自動的に回転させるためのものです。



ベースループのプロファイルビュー

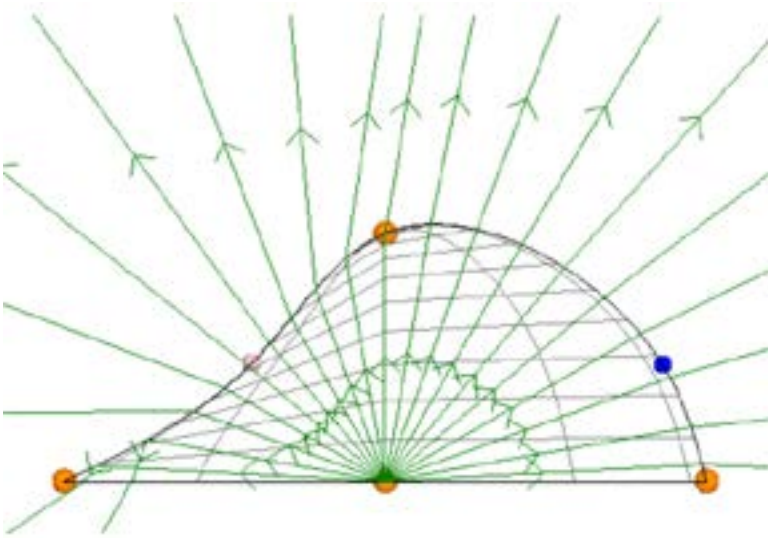


セクショナルプロファイルのプロファイルビュー

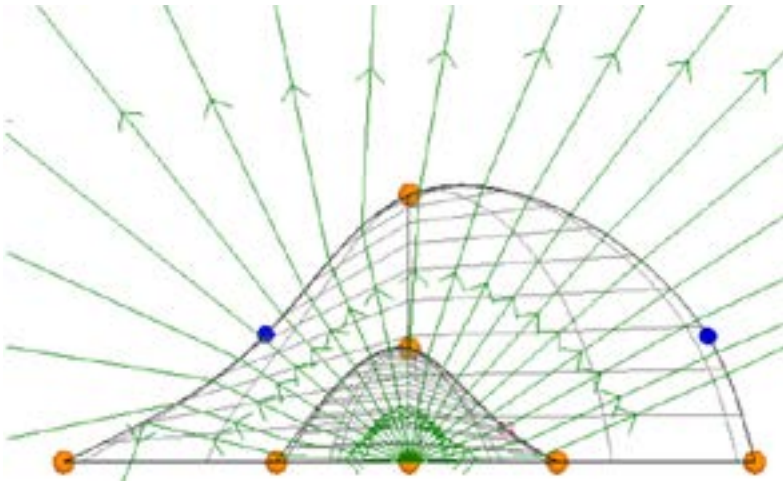


制御点を移動中、「**光線を表示**」オプションがチェックされていれば、リアルタイムで光線追跡が同時に更新されます。同時に選択できる自由曲面（放射タイプ）は最大2つまでです。

1つのサーフェスを選択した場合：

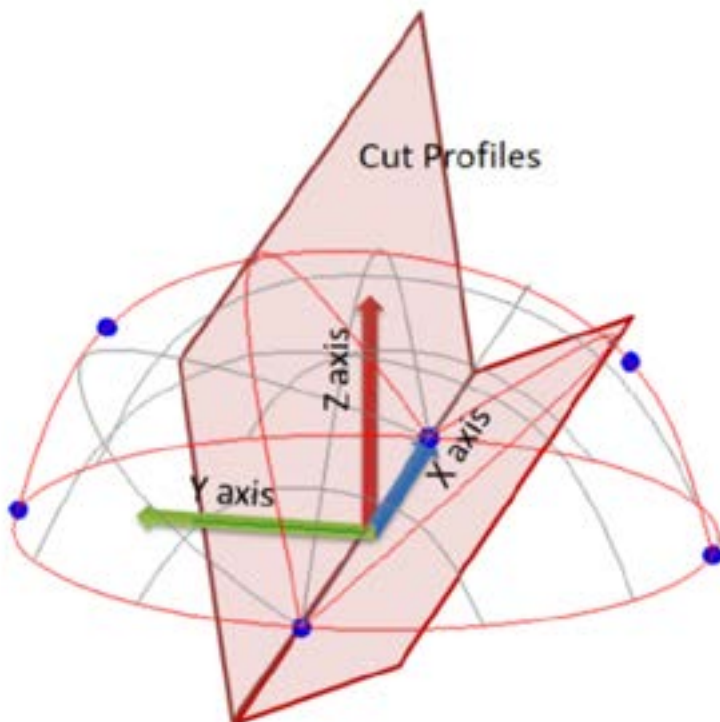
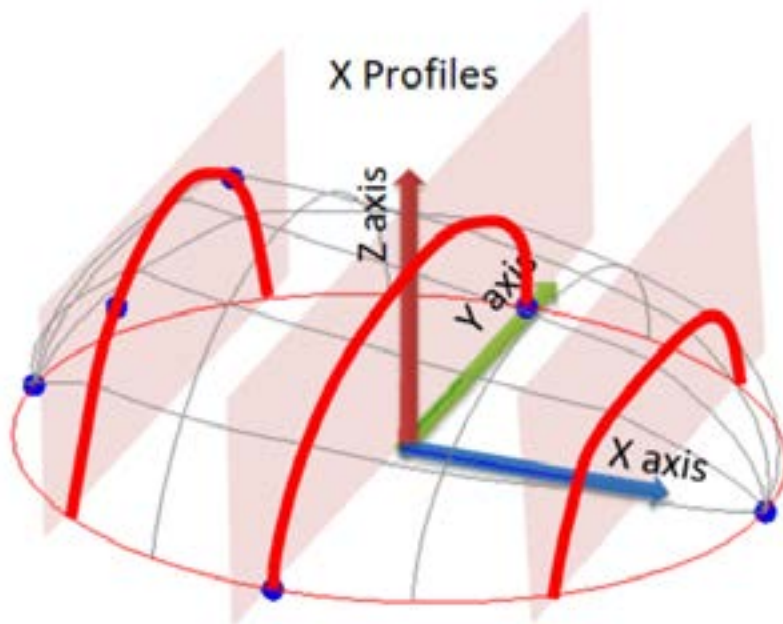


2つのサーフェスを選択した場合：



Type A サーフェス：自由曲面 - スライスタイプ

スライスタイプの自由曲面は、1つのベースループと複数の断面プロファイル（1つ以上）によって構築されました。プロファイルは、x プロファイルとカットプロファイルの2つのタイプに分類されます。x プロファイルはX 軸に垂直な平面断面を定義し、カットプロファイルはxy 平面に対する傾斜した平面断面です。



ベースループは平面上にあり、全ての断面プロファイルが接合する頂点は原点を通る垂直線上にあります。これにより、自由曲面スライスタイプが占有できるのは半球状の空間のみに制限されます。ベースループ上でマウスを右クリックすると、制御点の追加やX プロファイルの挿入が可能です。ベースループの対称性オプションは、制御点が対称的に挿入されるか（象限、X 偶数、Y 偶数）、独立して挿入されるか（なし）を制御することにご注意ください。

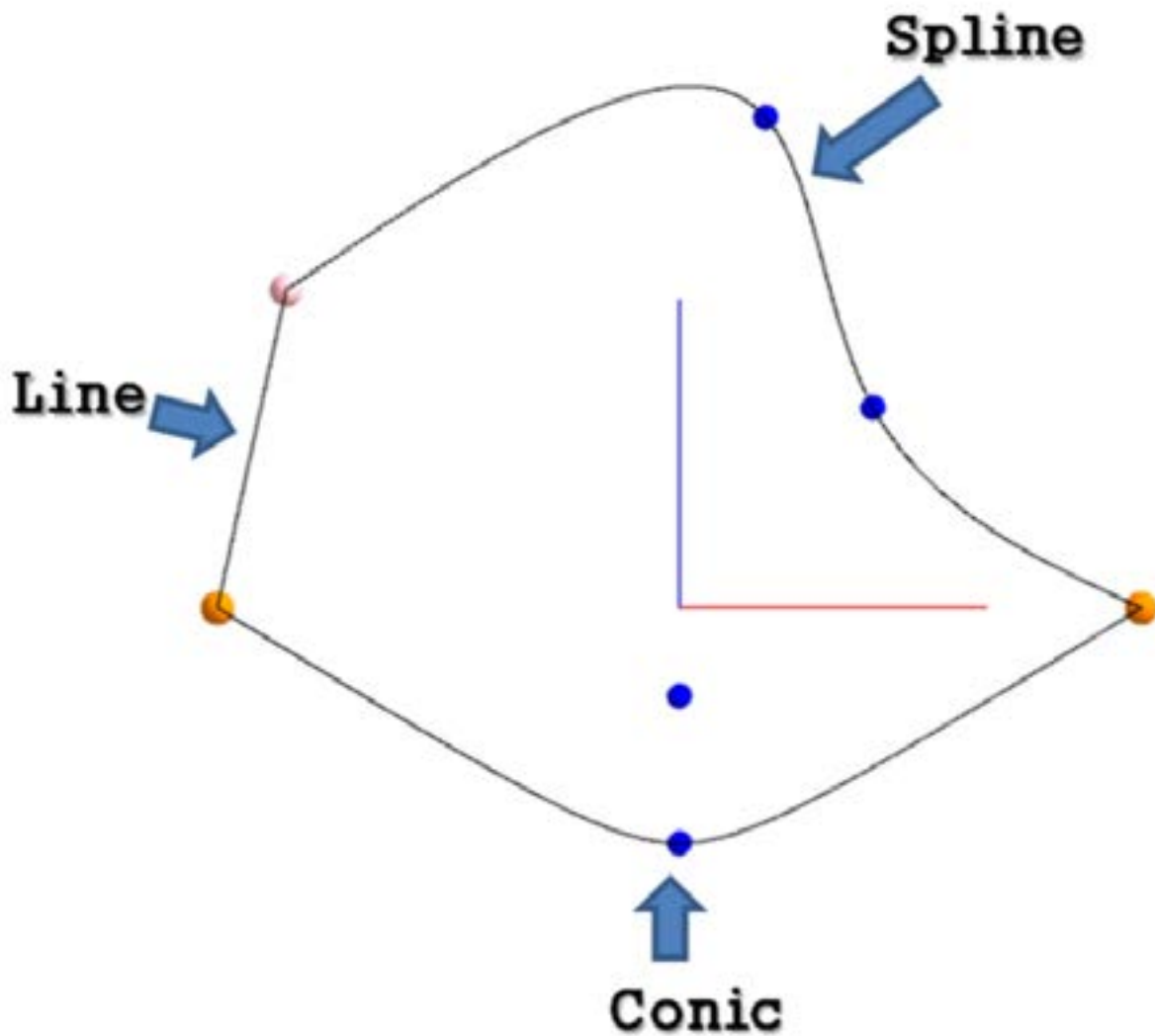
X プロファイルとカットプロファイルの対称性オプションは、挿入される制御点がペアになるかどうかを決定します。

タイプ B サーフェス：非対称プロファイル

閉じた二次元プロファイルと開いた二次元プロファイルはどちらもタイプ B サーフェスです。インタラクティブ最適化では、3つのサブクラスが定義されています。

「非対称プロファイルは、セグメント点によって結合された複数のセグメント（3つ以上）で構成されます。非対称プロファイルの形成に制限はありませんが、セグメントが交差することは許可されておらず、プロファイルは Y-Z 平面内になければなりません。

「各セグメントは異なるタイプであっても構いません。線、スプライン、またはコニック（半径とコニック定数を含む）を選択できます。



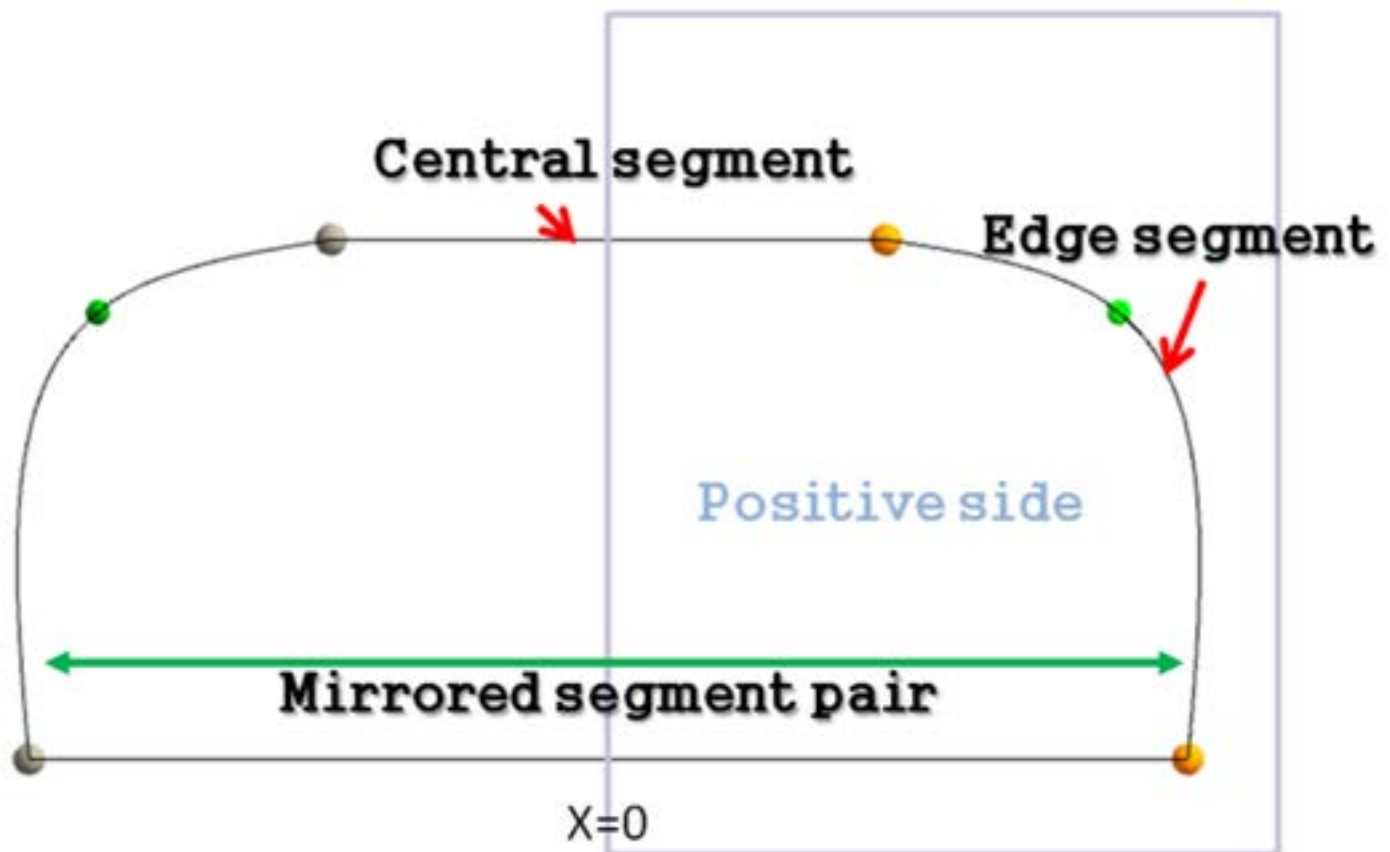
タイプ B サーフェス：対称プロファイル

閉じられた 2 次元プロファイルと開いた 2 次元プロファイルはどちらもタイプ B サーフェスです。インタラクティブオプティマイザには 3 つのサブクラスが定義されています。

レンズ設計では、放射状に対称なレンズ面がしばしば望まれます。プロファイルビューでは、プロファイルは Z 軸を中心とした回転対称性を持っています。しかし、上記で述べた非対称プロファイルでこれを実現するのは効率的ではありません。この場合は、対称プロファイルを使用してください。

対称プロファイルは、Z 軸が対称軸であると仮定します。

セグメントがスプラインであり、それが中央セグメント（つまり、セグメントがミラーリング基準線 ($X=0$) を横切る場合）である場合、制御点を追加すると、ミラー位置 ($x=-x$) にも別のミラーポイントが追加されます。これにより、結果のスプラインが対称であることが保証されます。また、スプラインセグメントが中央セグメントではなく、サイドセグメント、つまり開始セグメントと終了セグメントの両方の点が基準軸の同じ側にある場合、反対側にミラーリングされたセグメントが作成されます。ただし、スプラインを生成するルールは、スプライン曲線が開始セグメントポイントから終了セグメントポイントまで補間されることであり、エッジセグメントのミラー側は反対方向になるため、生成されるスプライン曲線は軸 $x=0$ に関してミラーリングされません。インタラクティブオプティマイザでは、スプラインの生成は正の側のセグメントの設定に従い、すべてのデータポイントの x 値の符号を反転させて、負の側に正しいミラーリングされた曲線を作成します。

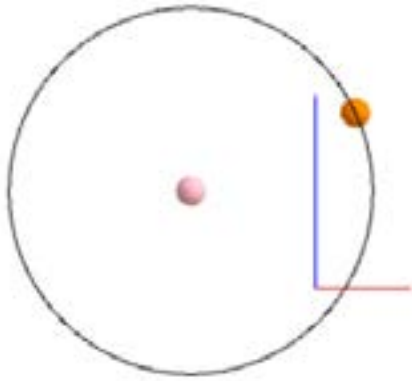


タイプ B サーフェス：楕円プロファイル

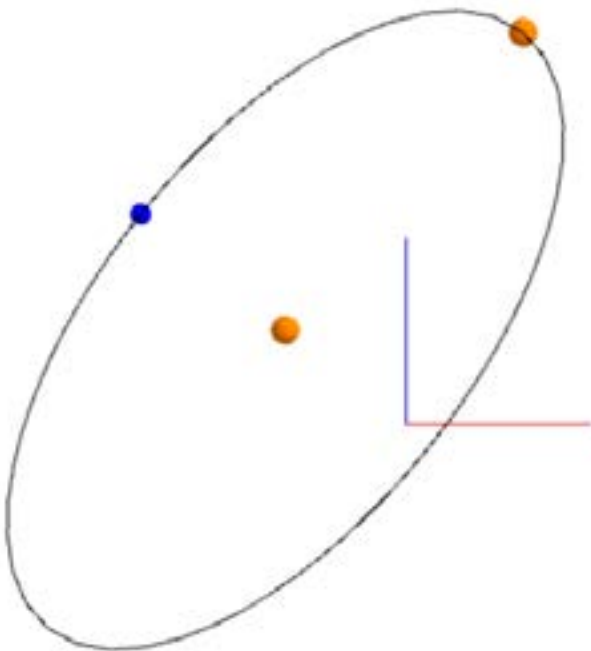
閉じたプロファイルと二次元プロファイルはどちらもタイプ B サーフェスです。インタラクティブオブティマイザでは、3つのサブクラスが定義されています。

楕円プロファイルは、円形および楕円形のプロファイル用に設計されています。

新しい楕円プロファイルを作成すると、プロファイルの中央と円弧上に2つの制御点が表示されます。ループの中央の点はプロファイルの位置を決定し、円弧上のもう一方の点は半径を制御します。

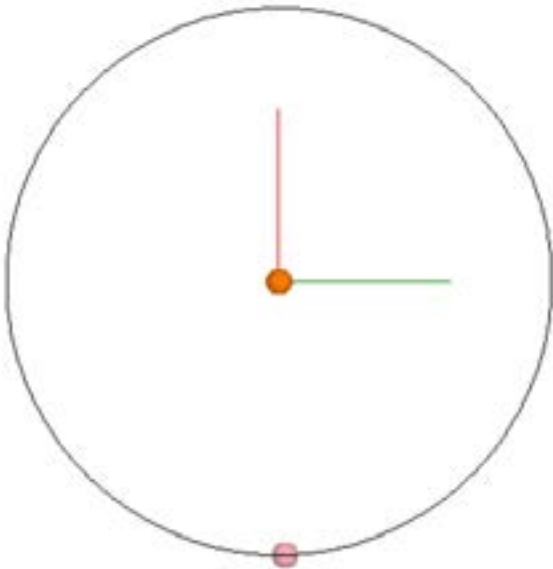


楕円にするには、右クリックして円上に新しい制御点を追加します。新しい制御点は楕円のもう一方の軸に沿って表示され、結果として得られるループは楕円になります。楕円の縦横比とサイズを変更するには、円弧の制御点を中央に向かって、または中央から離れるようにドラッグします。楕円を回転させるには、最初の円弧の制御点を中央の周りにドラッグします。

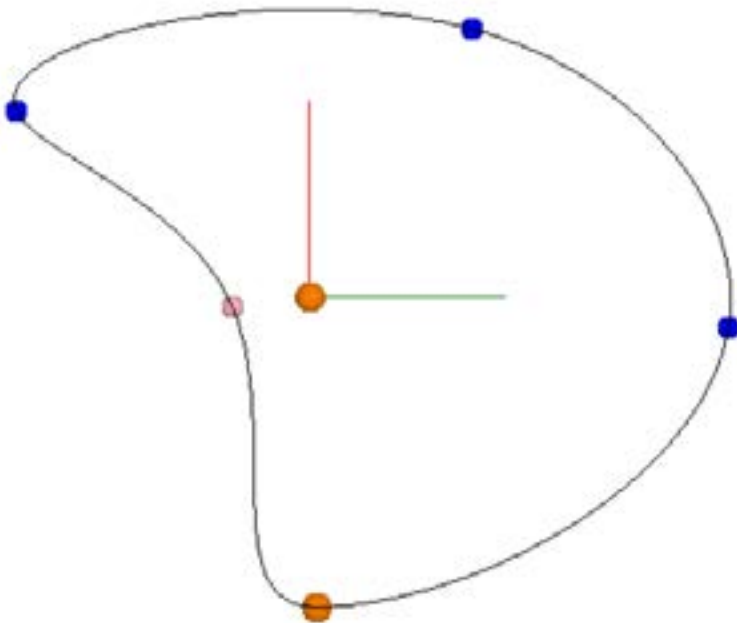


タイプ B サーフェス：自由形状ループ

自由形状ループは閉じた形状のプロファイルです。作成を開始すると、中心のコントロールポイントを原点とする円が形成されます。プロファイル上の唯一のコントロールポイントは、その半径を調整するためのものです。

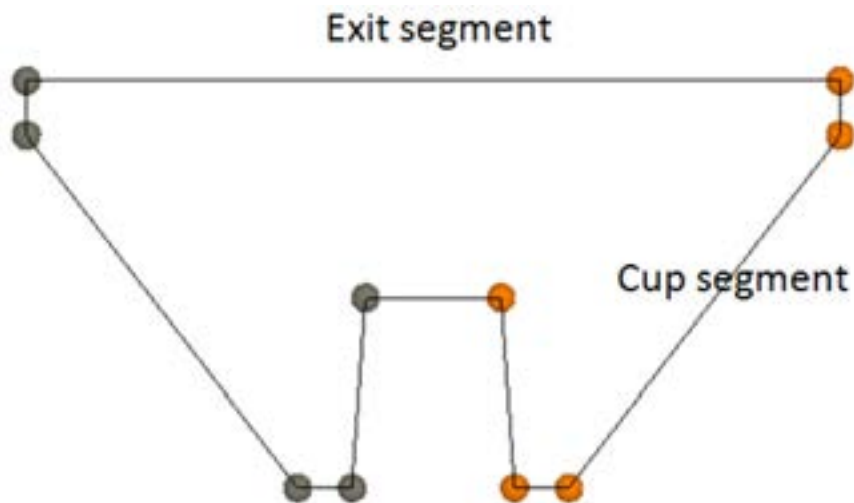


制御点がさらに追加されると、中央の制御点の位置がローカルオブジェクト座標系の原点となり、その形状はすべての制御点によって形成されます。



タイプ B サーフェス：全反射レンズ形状

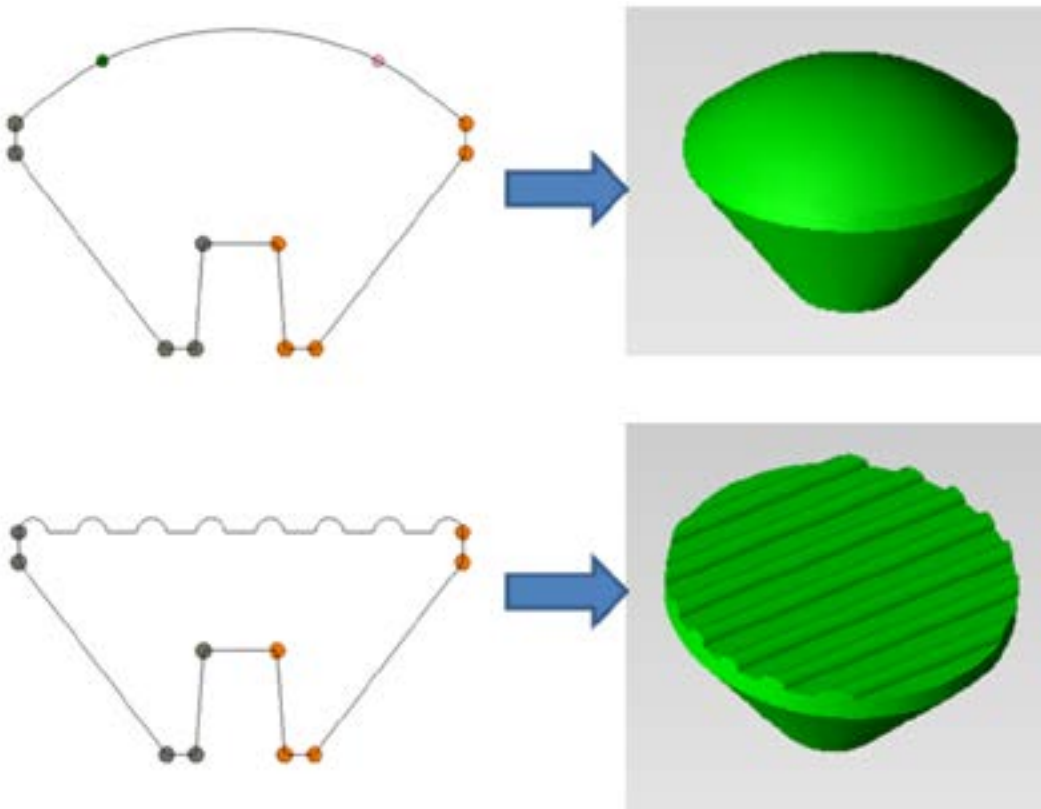
全反射レンズ形状は、全反射光学系のためにあらかじめ定義された対称な形状です。したがって、対称形状を持つすべての特性を備えています。



10 個のセグメントポイントはすべて削除できませんが、セグメントの種類は変更できます。

TIR レンズのプロファイルにおいて最も重要なセグメントは、出射セグメントとカップセグメントです。なぜなら、TIR レンズオブジェクトを作成する際に、これらの 2 つのセグメントは特別に扱われるからです。

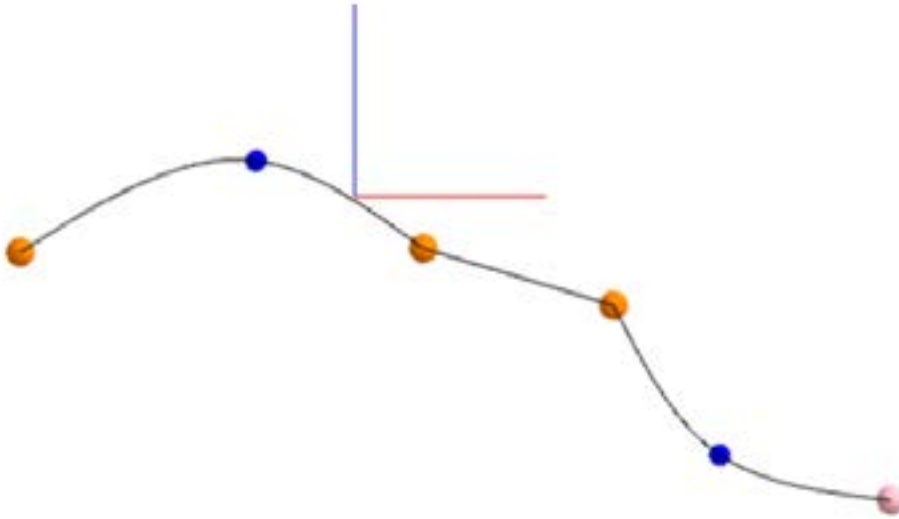
TIR レンズは、LED のような拡張ソースから放出されるビームを狭めるためによく使用されます。車両のヘッドランプへの応用を考慮すると、出射面に一連のシリンドリカルレンズを配置することは、出射ビームを X 軸と Y 軸で再成形するための一般的なアプローチです。設計要件を考慮すると、出射セグメントが周期セグメントとして選択された場合、出射セグメントの操作は回転ではなく押し出しになります。



タイプ C サーフェス：2D パス

タイプ C サーフェスは、開いた形状（オープンプロファイル）を使用して作成されます。

インタラクティブオプティマイザでサポートされているオープンプロファイルは、2D パスです。これは複数のセグメントを含みますが、最初のセグメントと最後のセグメントが必ずしも結合している必要はありません。2D パスの場合、すべてのセグメントと制御点は X-Z 平面上に存在する必要があります。これが 2D パスという名前の由来です。

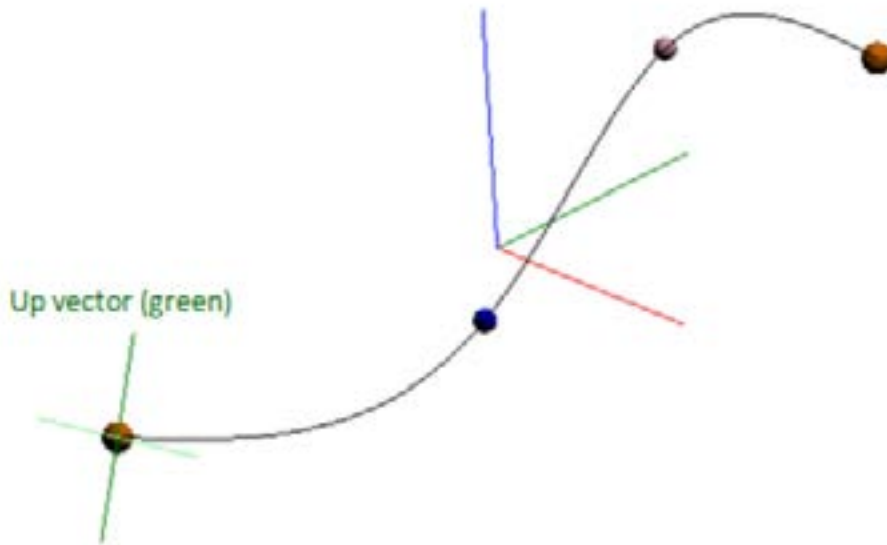


2D パスの主な目的は、「スイープ」機能を使用してオブジェクトを作成する際のスイープ経路を指定することです。

Type C サーフェス：3D パス

Type C サーフェスは、開いたプロファイルとして構築されます。

インタラクティブオブティマイザは、2D パスの他に、2D パスの高度なバージョンと認識できる、より柔軟なパスである 3D パスを提供します。2D パスと同様に、3D パスも複数のセグメントを含み、最初のセグメントと最後のセグメントが結合している必要はありません。しかし、2D パスとは異なり、すべてのセグメントと制御点が任意の平面上にある必要はなく、3D 空間内を自由に移動できます。3D パスの性質上、ユーザーが追加のプロパティ（アップベクトル）を定義する必要があります。アップベクトルは、開始点におけるパスの方向に対して垂直である必要があり、プロパティダイアログに入力された際に、この基準を満たすように自動的に修正されます。

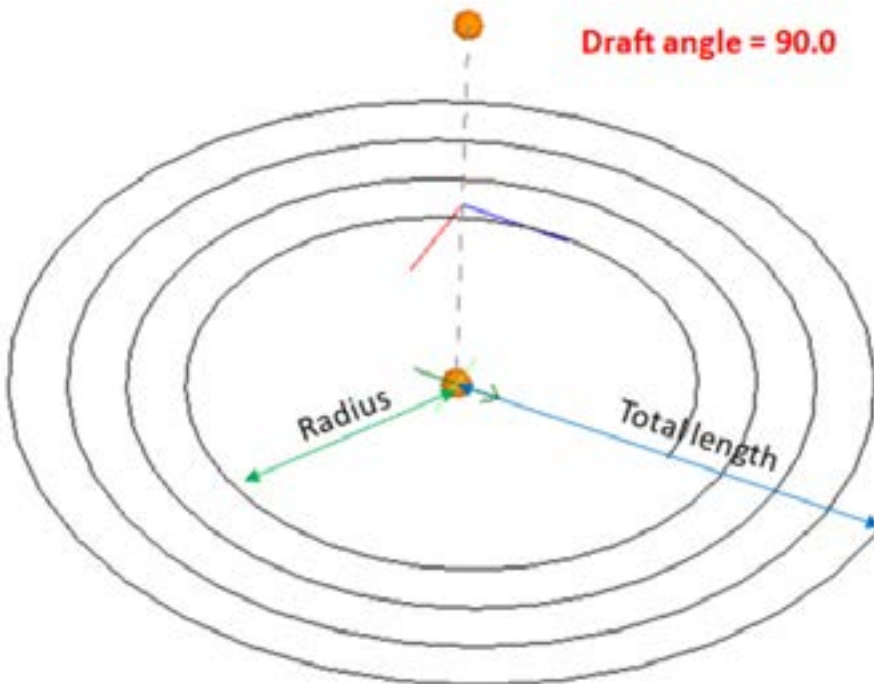
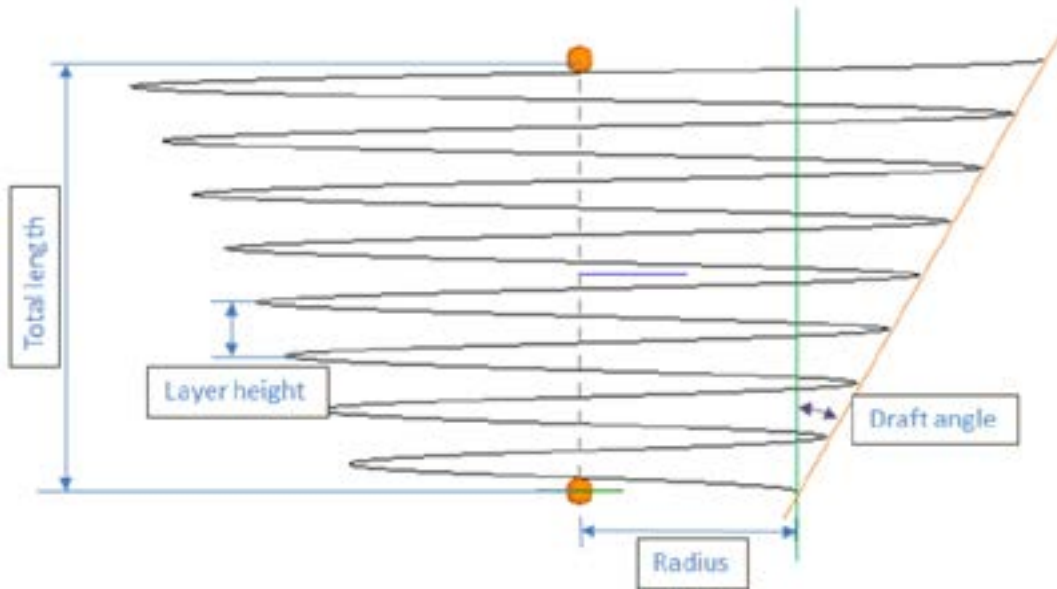


3D パスの主な目的は、「スイープ」メソッドを使用してオブジェクトを作成する際のスイープ経路を指定することです。

Type C サーフェス：螺旋パス

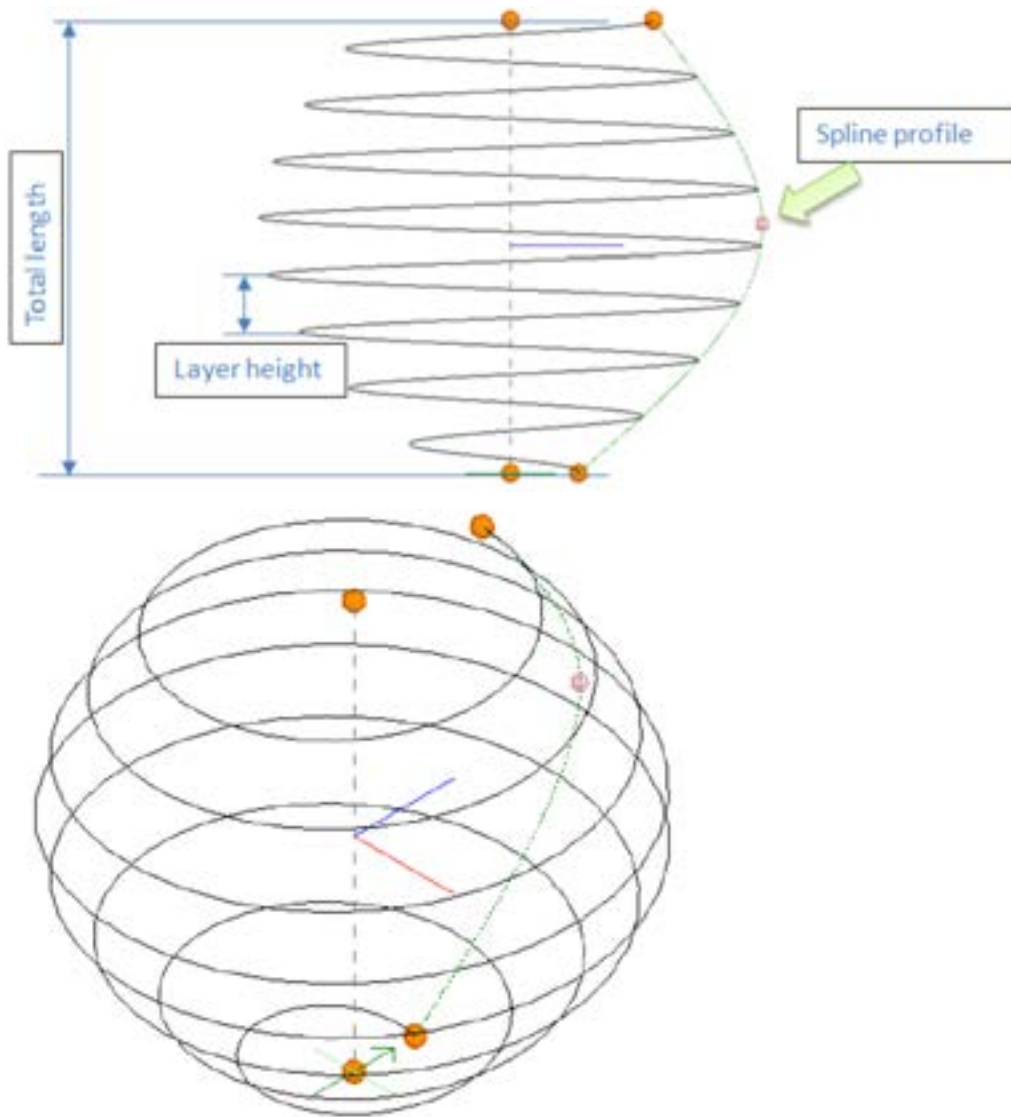
Type C サーフェスは、開いたプロファイルを使用して構築されます。

螺旋パスは、らせん状の形状です。2つの終点は、螺旋の軸とその長さを定義するために使用されます。螺旋の枝間の距離は固定されており、「レイヤー高さ」係数として設定されます。「半径」係数は、螺旋方程式を構成する最初の半径です。「抜き勾配」は、螺旋が徐々に成長または減少するためのものです。抜き勾配が 90.0 の場合、結果として得られる形状は平面上にあります。



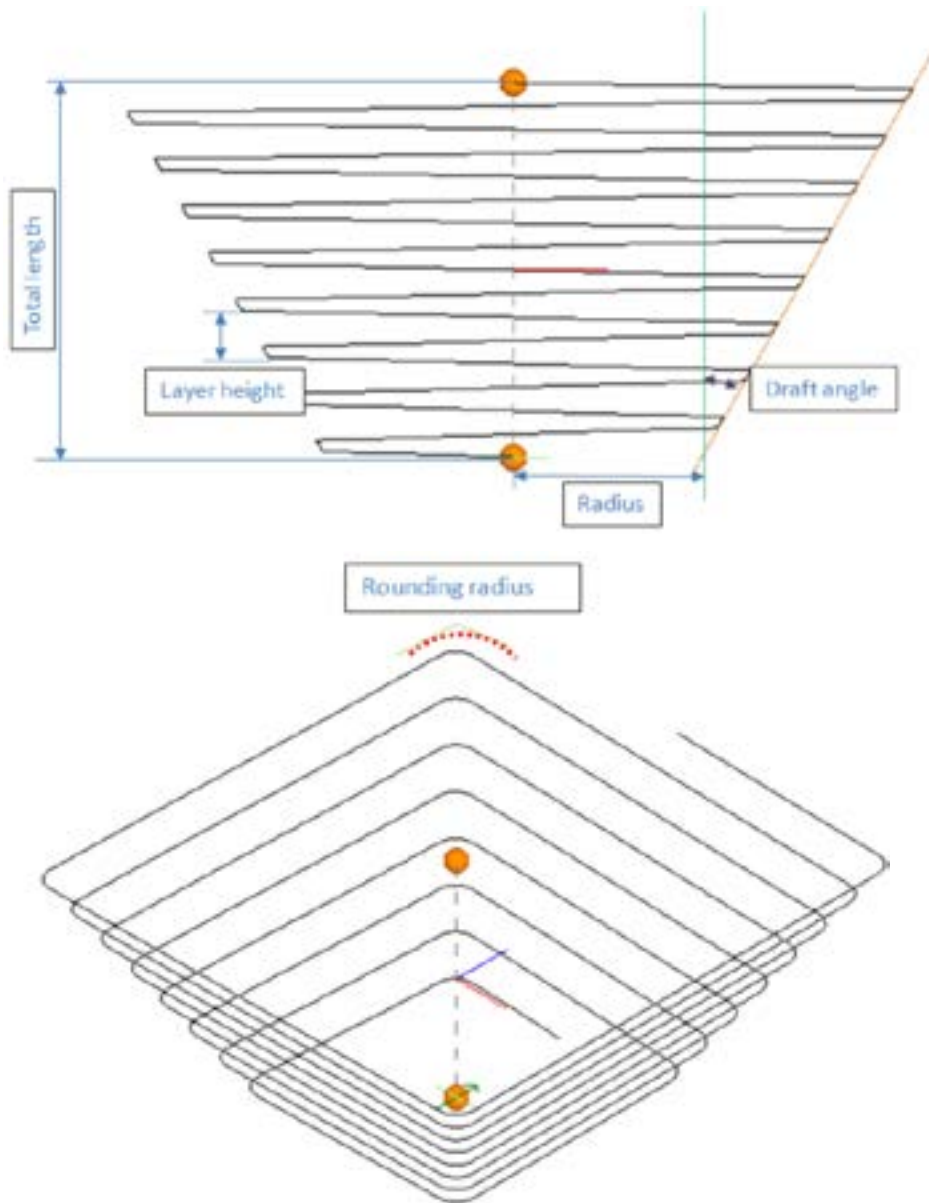
Cタイプサーフェス：自由螺旋パス

自由螺旋パスは、螺旋パスの上位バージョンです。スプラインセグメントを制御して、螺旋曲線の輪郭形状を定義できます。螺旋形状は入力プロファイルによって決定されるため、自由螺旋パスでは半径とドラフト角の2つの係数は削除されています。



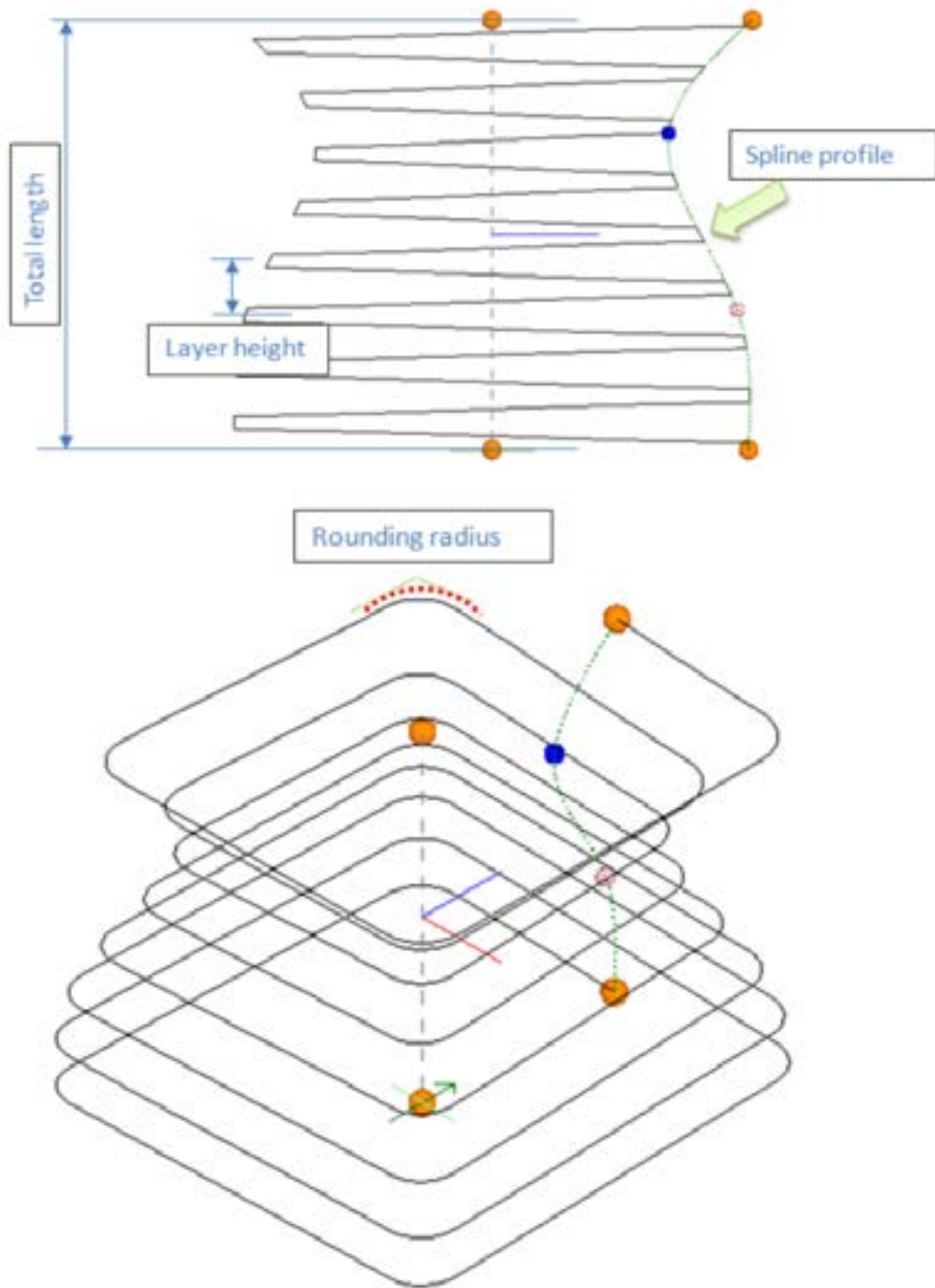
C タイプサーフェス：矩形螺旋パス

矩形螺旋は、螺旋パスと類似の係数を持ちますが、追加のパラメータが一つあります - 丸め半径です。矩形螺旋パスに沿って断面形状をソリッドオブジェクトとしてスイープする場合、丸めは必須で、さもないと TracePro でジオメトリを構築する際に自己交差エラーが警告されます。



タイプC サーフェス：自由正方スパイラルパス

自由正方スパイラルパスと正方スパイラルパスの差は、自由らせんパスとらせんパスの差と同様です。自由正方スパイラルパスは、プロフィールを任意に定義できる自由度を提供します。

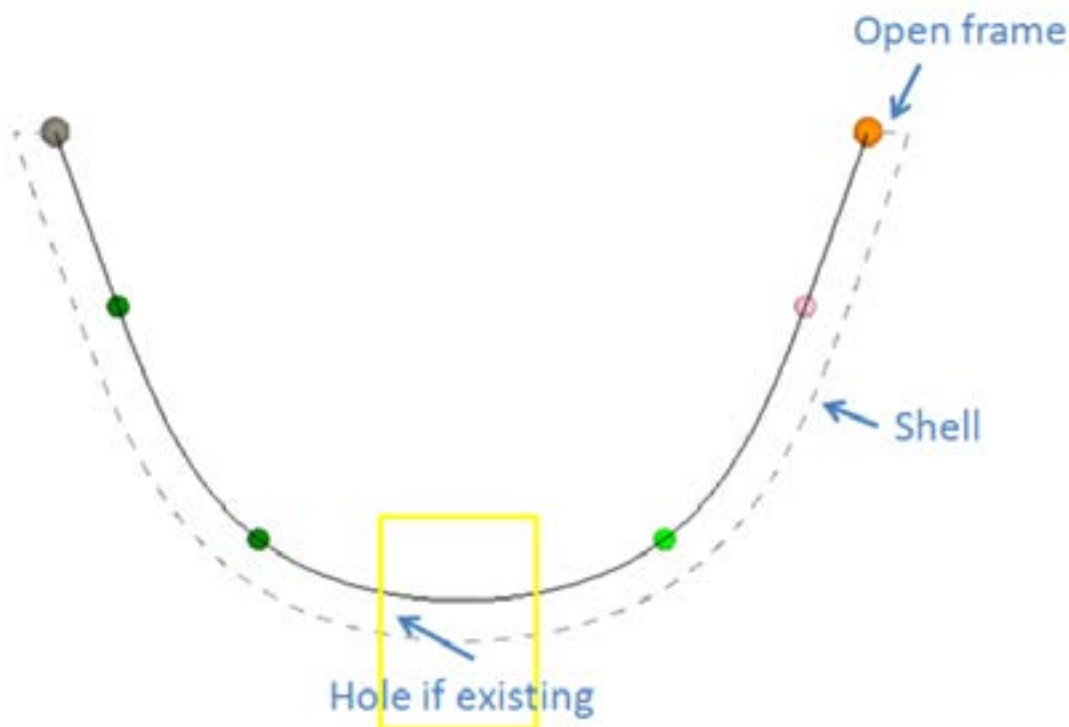


D タイプサーフェス：スプライン反射鏡

ほとんどの反射鏡設計では、単一のプロファイルのみが必要です。通常の反射鏡設計における冗長な操作を防ぐため、D タイプサーフェスはそのような場合に設計されています。

「スプライン反射鏡は、単一のスプラインセグメントを使用して反射鏡の形状を制御します。サーフェスプロパティでは、反射鏡の厚さを指定できるため、TracePro にエクスポートする際に、ソリッドオブジェクトとして構築できます。他のタイプのサーフェスのスプラインセグメントと同様に、より多くの制御点を自由に追加して、設計目標を満たすより複雑なプロファイルを実現できます。

すべての反射鏡は、オブジェクト - 反射鏡と連携する必要があります。



破線は、スプラインプロファイルから厚さの値だけ離れた位置に自動的に生成されます。

反射板の厚さは、スプライン反射板に付随する唯一のパラメータではありません。他の 3 つの構成要素のサーフェス特性も指定できます。

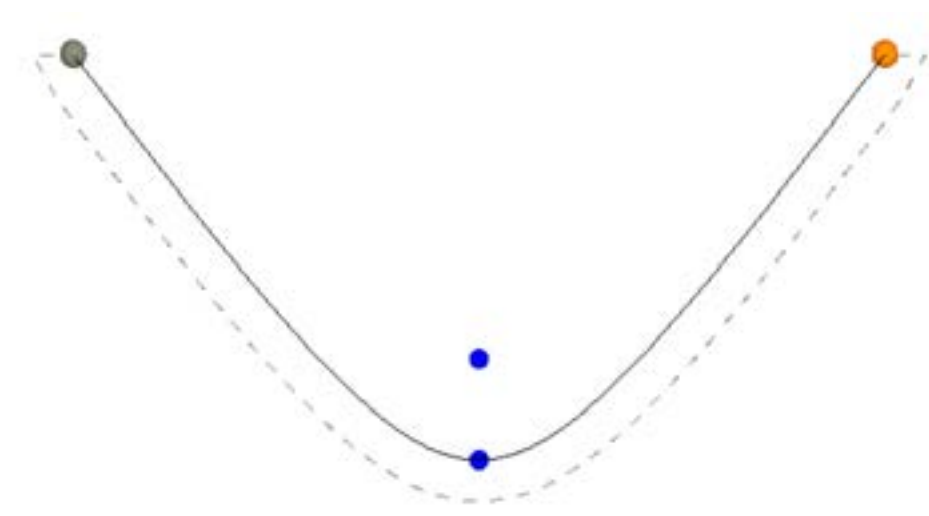
Description	Value
Name	Spline_reflecto...
Surface type	Spline Reflecto...
Surface ID	1
Thickness	1
Shell surf. catalog	
Shell surf. property	
Open surf. catalog	
Open surf. property	
Hole surf. catalog	
Hole surf. property	

しかし、反射面のサーフェス特性そのものは、他のサーフェスタイプとの操作概念との一貫性を持たせるために、セグメントプロパティ内で定義されています。

D タイプサーフェス：コーンリフレクタ

スプライン反射鏡は柔軟性がありますが、その柔軟性ゆえに、放物面、楕円面、球面反射鏡のようなパラメトリックな形状を望む場合には困難が生じます。

コーンリフレクタは、この問題を解決するためにあります。



円錐リフレクターはスプラインリフレクターと似ていますが、円錐セグメントを使用する点が異なります。円錐パラメータ（CC と CV）は、2つの制御点をドラッグするか、手動で値を入力することによって変更できます。また、円錐方程式から導かれる特定の種類のパラメトリック曲線が必要な場合は、円錐の種類を固定することもできます。

セグメント

プロファイルの作成にセグメントを使用しているサーフェスの場合、サポートされているセグメントタイプは以下の通りです。ただし、各サーフェスの特性上、有効なセグメントはそれぞれ異なります。

- [直線](#)
- [スプライン](#)
- [円錐曲線](#)
- [周期](#)
- [カスタムフレネル](#)

曲線タイプのセグメント、すなわちスプラインと円錐曲線については、フレネル化がセグメントを管理するための機能として提供されています。これは曲線をフレネルプロファイルに変換するもので、詳細については[フレネル化](#)のページをご覧ください。

対称プロファイル内の中央のスプラインセグメントについては、非球面フィッティングツールを使用してスプライン曲線の非球面パラメータを取得できます。詳細については、[非球面フィッティング](#)のページをご覧ください。

セグメント - ライン

説明:

ラインは、サポートされているセグメントの中で最も基本的なタイプです。線セグメントには、さらに付加的なプロパティはありません。

アクション:

なし。

プロパティ:

サーフェスカタログ: TracePro のサーフェスプロパティデータベースから、サーフェスプロパティのカタログを選択します。この設定はインタラクティブオプティマイザ内では何の影響も与えず、ジオメトリを TracePro にエクスポートする際にのみ使用されます。

サーフェスプロパティ: 上記で選択したカタログから、TracePro データベース内のプロパティを選択します。この設定はインタラクティブオプティマイザ内では何の影響も与えず、ジオメトリを TracePro にエクスポートする際にのみ使用されます。

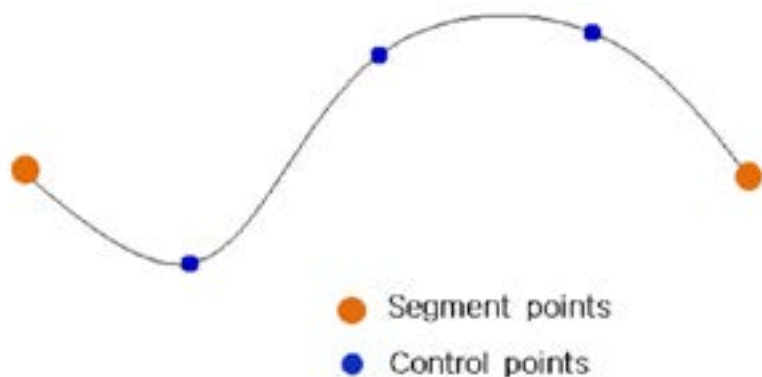
反射 ?: サーフェスリストダイアログでのリアルタイムレイトレーシングのために、選択したセグメントの反射フラグを設定 / リセットします。

吸収 ?: サーフェスリストダイアログでのリアルタイムレイトレーシングのために、選択したセグメントの吸収フラグを設定 / リセットします。「反射」フラグと「吸収」フラグは、どちらか一方のみを選択できます。

セグメント - スプライン

説明:

スプラインとは、曲線自体の中にある任意の数の制御点によって制御される、自由な形状の曲線です。



アクション

コントロールポイントを追加: マウスクールがある位置にコントロールポイントを挿入します。

コントロールポイントを削除: マウスクールがホバーしている選択されたコントロールポイントを削除します。

プロパティ:

サーフェスカタログ: TracePro のサーフェスプロパティデータベースからサーフェスプロパティのカタログを選択します。この設定はインタラクティブオプティマイザ内では何の影響も与えず、ジオメトリを TracePro にエクスポートする際にのみ使用されます。

サーフェスプロパティ: 上記で選択したカタログから TracePro データベース内のプロパティを選択します。この設定はインタラクティブオプティマイザ内では何の影響も与えず、ジオメトリを TracePro にエクスポートする際にのみ使用されます。

反射?: サーフェスリストダイアログでのリアルタイムレイトレーシングのために、選択されたセグメントの反射フラグを設定 / リセットします。

吸収?: サーフェスリストダイアログでのリアルタイムレイトレーシングのために、選択されたセグメントの吸収フラグを設定 / リセットします。反射フラグと吸収フラグはどちらか一方のみ選択可能です。

フレネル化?: 現在のスプラインセグメントに基づいてフレネル面を作成します。このオプションをオンにすると、プロパティウィンドウに以下の追加オプションが表示されます。

フレネル化手法: 選択可能なフレネル化手法には、「等高」モードと「等幅」モードの 2 種類があります。

等高: すべてのプリズムの高さが同じに保たれます。

等幅: すべてのプリズムの幅が同じに保たれます。

プリズムタイプ: プリズムの作成に「カーブ」または「トライアングル」を使用します。

プリズム高さ (等高モードのみ): 生成されるプリズムの高さ。

プリズム数 (等幅モードのみ): プリズムの数。

ベースラインタイプ: 生成されるフレネル面のベースラインを選択するオプションです。プリズムは、カーブプロファイルとその基準線（両端を結ぶ直線）の差によって作成され、選択されたベースライン上に配置されます。サポートされているベースラインタイプは、ライン、スプライン、コニックです。

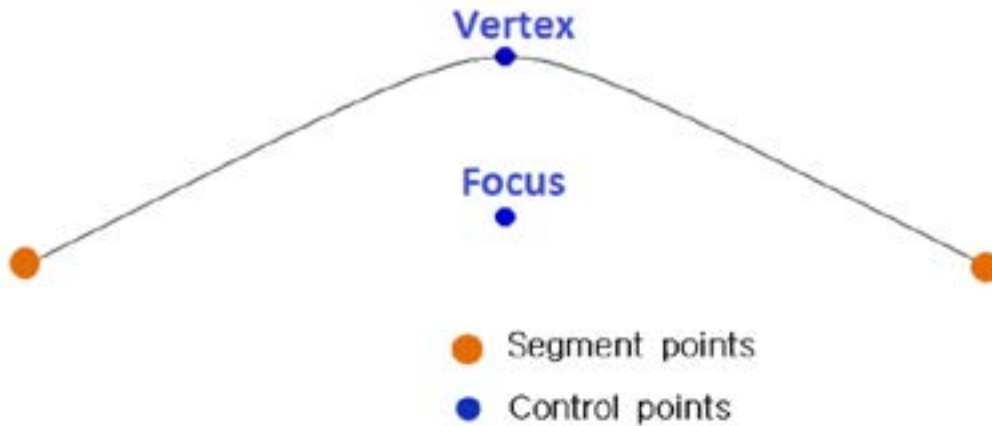
ファセット化?: カーブを連続した直線に変換します。

ファセット数: 生成するファセットの数。

セグメント - コーニック

説明：

もしプロファイルが円錐曲線項で記述できるならば、その曲線は円錐曲線セグメントです。円錐曲線セグメントは、通常の操作のために 2 つの制御点を持ちます。1 つはいわゆる焦点であり、もう 1 つは曲線の頂点です。これらの 2 つの制御点が指定されると、二次曲線の 2 つのパラメータ、すなわち曲率 (cv) と円錐曲線定数 (cc) が導き出され、逆もまた同様です。

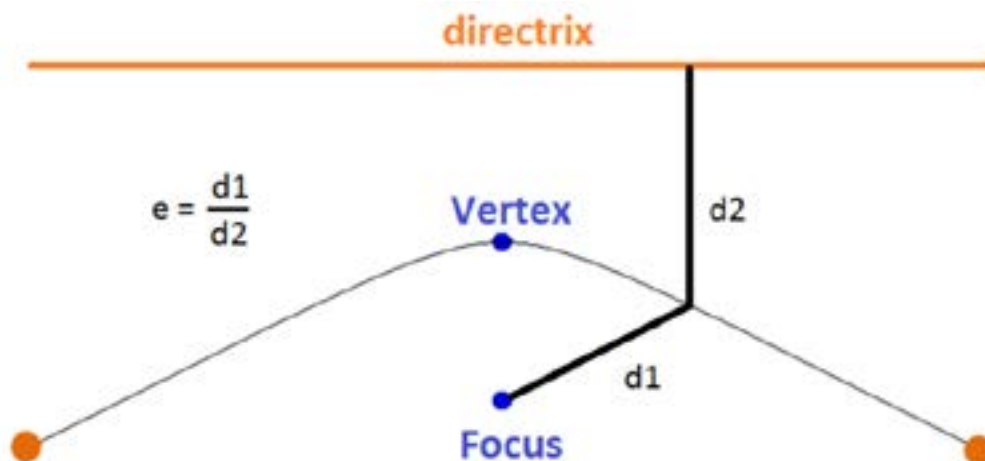


$$z = \frac{c v r^2}{1 + \sqrt{1 - c v^2 (c c + 1) r^2}} + \sum_j A S_j r^j$$

二次曲面のタイプは、以下の円錐定数の値によって決まります。

Surface Type	Conic constant (cc)
Circle	$cc = 0$
Ellipsoid	$-1 < cc < 0$
Paraboloid	$cc = -1$
Hyperboloid	$cc < -1$

基本的に、二次曲線は直線（準線と呼ばれる）と焦点を用いて定義できます。



あらゆる円錐曲線において、焦点から曲線上の点までの距離と、その点から準線に垂直に下ろした距離の

比は常に一定です。この比は離心率（e）と呼ばれます。言い換えれば、焦点までの距離が準線までの距離の離心率（e）倍に等しいすべての点の集合が円錐曲線となります。なお、円錐定数と離心率（e）の間には以下の関係があります。

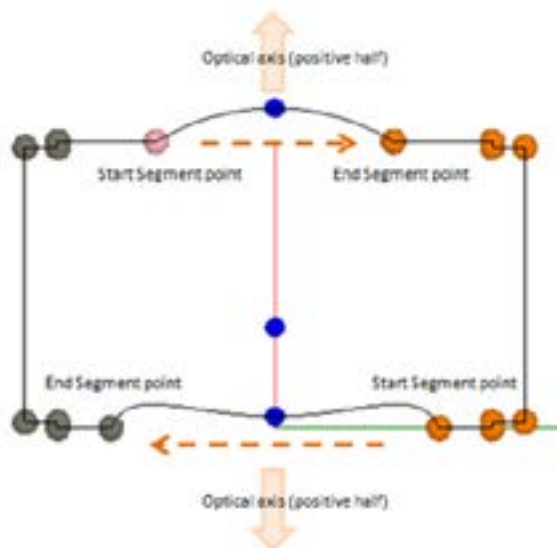
$$cc = -e^2$$

したがって、二次曲面のタイプは、以下のように離心率の値にも依存します。

Surface Type	Eccentricity (e)
Circle	$e = 0$
Ellipsoid	$0 < e < 1$
Paraboloid	$e = 1$
Hyperboloid	$e > 1$

係数の符号：

通常のレンズ設計ツールでは、光軸は反射による屈折を除き、固定された方向に定義されます。これにより、サーフェスの係数の符号は単純になります。しかし、インタラクティブオプティマイザでは、光軸という概念が存在しません。あるサーフェスに関連する正の半分は、セグメントの方向、つまり開始セグメント点から終了セグメント点への方向によって決定されます。



上記をレンズとして考えると、単に2つの面が反対方向の光軸を持っていることを示しており、設計されたレンズパラメータに対して係数の符号が反転することにつながります。ツールでレンズパラメータを入出力する際のこの混乱を解消するために、「Seg invert?」というプロパティが導入されました。このプロパティをチェックすると、指定されたセグメントの光軸の方向が反転するため、表示される曲率や非球面項は設計時と同じように一貫したものになります。

アクション：

なし。

プロパティ：

コニックタイプ: ここでは合計4つのタイプが機能しており、コニック曲線の3つの退化形を含みます。円形はコニック定数が0.0、放物線はコニック定数が-1.0、そして楕円形です。

コニック定数: 方程式で使用されるコニック定数（CC）。

曲率: 方程式で使用される曲線の曲率（CV）。

非球面項を使用しますか？：非球面項を使用するかどうかを選択します。

非球面タイプ：**「標準」、「奇数」、「偶数」、「一般」の中から選択します。

非球面項数：** 方程式で使用する非球面項の数を指定します。

rⁿ：各項の非球面係数を入力します。

サーフェスカタログ：TracePro のサーフェスプロパティデータベースからサーフェスプロパティのカタログを選択します。この設定はインタラクティブオブティマイザには影響せず、ジオメトリを TracePro にエクスポートする際にのみ使用されます。

サーフェスプロパティ：上記で選択したカタログから、TracePro データベース内のプロパティを選択します。この設定はインタラクティブオブティマイザには影響せず、ジオメトリを TracePro にエクスポートする際にのみ使用されます。

反射性？：サーフェスリストダイアログでのリアルタイムレイトレーシングのために、選択したセグメントの反射フラグを設定 / リセットします。

吸収性？：サーフェスリストダイアログでのリアルタイムレイトレーシングのために、選択したセグメントの吸収フラグを設定 / リセットします。反射フラグと吸収フラグはどちらか一方のみを選択できます。

フレネル化？：現在のスプラインセグメントに基づいてフレネル面を作成します。このオプションをオンにすると、プロパティウィンドウに以下の追加オプションが表示されます。

フレネル化手法：選択可能なフレネル化手法には、「等高線」モードと「等幅」モードの 2 種類があります。

等高線：すべてのプリズムの高さが同じに保たれます。

等幅：すべてのプリズムの幅が同じに保たれます。

プリズムタイプ：プリズムを作成するために「カーブ」または「三角形」を使用します。

プリズム高さ（等高線モードのみ）：生成されるプリズムの高さ。

プリズム数（等幅モードのみ）：* プリズムの数。

ベースラインタイプ：生成されるフレネル面のベースラインを選択するオプションです。プリズムは、カーブプロファイルとその基準線（両端を結ぶ直線）の差によって作成され、選択されたベースライン上に配置されます。サポートされているベースラインタイプは、直線、スプライン、およびコニックです。

ファセット化？：カーブを連続した直線に変換します。

ファセット数：生成するファセットの数。

セグメント反転？：レンズ面の光軸の方向を反転させます。

セグメント - 周期

説明:

周期セグメントは、その上にいくつかの種類の微細構造を作成する機能を持っています。また、スプラインとしても機能するため、制御点を追加することで、周期パターンの基準となる曲線状のプロファイルを作成できます。

アクション:

制御点を追加: マウ斯卡ーソルがある位置に制御点を挿入します。

制御点を削除: マウ斯卡ーソルが重なっている選択された制御点を削除します。

プロパティ:

ジオメトリタイプ:

なし (None): 通常のスプラインとして機能します。

円 (Circle):



ピッチ: 単一パターンの長さ

半径: パターンの作成に使用する円の半径

高さ: 基準面からの円の高さ

三角形:



ピッチ: 単一パターンの長さ。

幅: 三角形の幅。

高さ: 基準面からの三角形の高さ。

ピーク位置: 分数による測定でピークがどこに位置するか。0.5 の場合、三角形は二等辺三角形になります。0.0 または 1.0 の場合、それぞれ左または右に向けた直角三角形が作成されます。

TriCircle:



Pitch: 単一パターンの長さ。

Central radius: Tricircle（トライサークル）パターンは、連続する 3 つの円によって形成され、両側の 2 つの円は同一ですが、中央の円とは異なる場合があります。ここでは、中央の円の半径を入力します。

Marginal radius: 両側の 2 つの円の半径。

Trapezium: 台形



ピッチ: 単一パターンの長さ。

幅: 台形の底辺の幅。

高さ: 基準面からの台形の高さ。

角度 1: 左側の側壁の垂直方向の角度（度）。

角度 2: 右側の側壁の垂直方向の角度（度）。

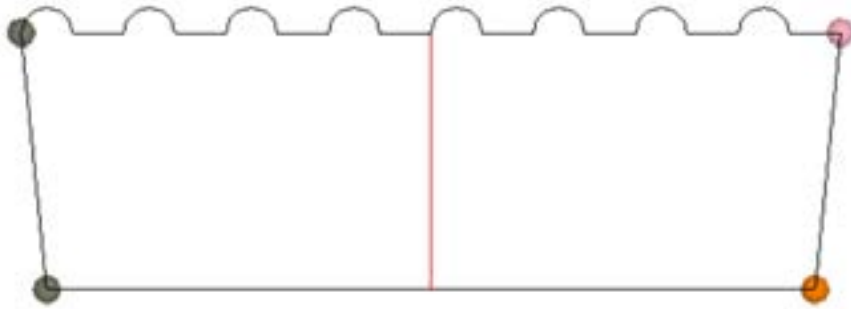
General properties（一般プロパティ）:

Reflective? (反射性 ?): サーフェスリストダイアログにおけるリアルタイムレイトレーシングのために、選択されたセグメントの反射フラグを設定 / リセットします。

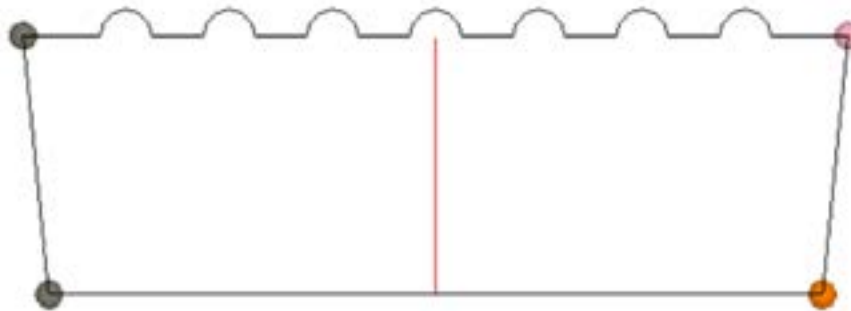
Absorptive? (吸収性 ?): サーフェスリストダイアログにおけるリアルタイムレイトレーシングのために、選択されたセグメントの吸収フラグを設定 / リセットします。「Reflective (反射性)」と「Absorptive (吸収性)」フラグはどちらか一方のみ選択可能です。

Center feature options (中心フィーチャーオプション) :

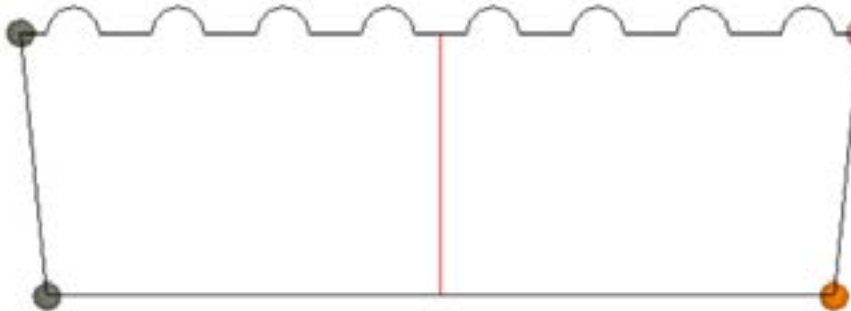
None (なし): フィーチャー生成は、開始セグメントから終了セグメントの点まで順番に行われます。



機能：フィーチャーをセグメントの中央に強制的に配置。



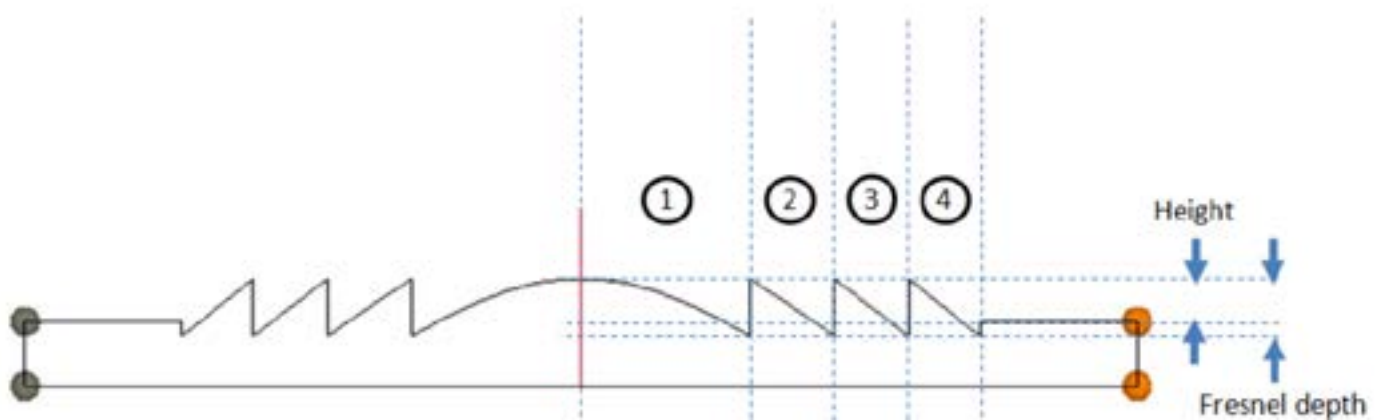
Space: フィーチャーの間隔が中央になるように配置



「中心フィーチャー」のオプションが「None」の場合、シフトは、フィーチャーの生成が開始セグメントの始点からどれだけ離れて始まるかの量。

セグメント - カスタムフレネル

あらかじめ記述された半径と円錐係数のテーブルを用いて、フレネル面の各セグメントを個別に制御できます。カスタムフレネルセグメントでは、フレネルセグメントの数を入力し、必要に応じて各セグメントに半径の値と円錐定数を割り当てることができます。



セグメントの頂点から測定されたサグ方程式は以下の通りです。

$$z = \frac{ch^2}{1 + \sqrt{1 - (k+1)c^2h^2}}$$
$$c = \frac{1}{r}$$

z はサグ、 h は中心からの距離、 c は曲率、そして k はコニック定数です。

アクション:

なし

高さ: ベースとなる線分から中央の頂点までの距離。

フレネル深さ: フレネルプリズムの高さ。

Rn: n 番目のフレネルセグメントの半径の値。

Kn: n 番目のフレネルセグメントの円錐定数。

反射 ?: サーフェスリストダイアログのリアルタイムレイトレーシングにおいて、選択されたセグメントの反射フラグを設定 / リセットします。

吸収 ?: サーフェスリストダイアログのリアルタイムレイトレーシングにおいて、選択されたセグメントの吸収フラグを設定 / リセットします。「反射」と「吸収」フラグはどちらか一方のみ選択可能です。

フレネル化

説明：

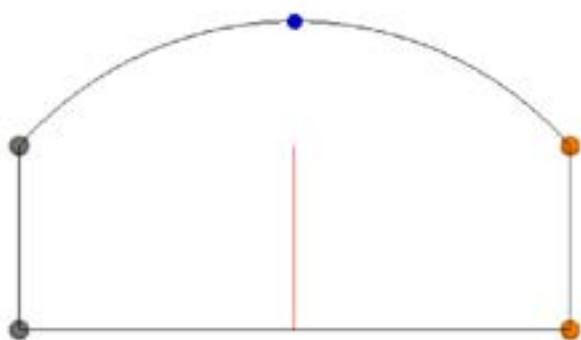
フレネル化とは、曲線をフレネル型のプロファイルに変換することです。この機能は、スプラインセグメントやコニックセグメントなど、その機能を持つセグメントのプロパティエディタに表示されるオプションです。

フレネル化のプロセスは、単純にプロファイルを両端のセグメント端点を結ぶ基準線に向かって「引き下げる」ことです。異なる要件に対応するため、等高モードと等幅モードの2つのフレネル化手法が利用可能です。

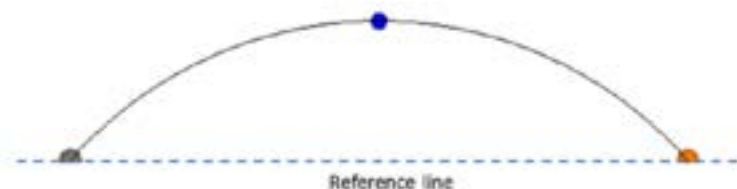
等高モード：

このモードは、すべてのプリズムの高さを同じに保ちます。

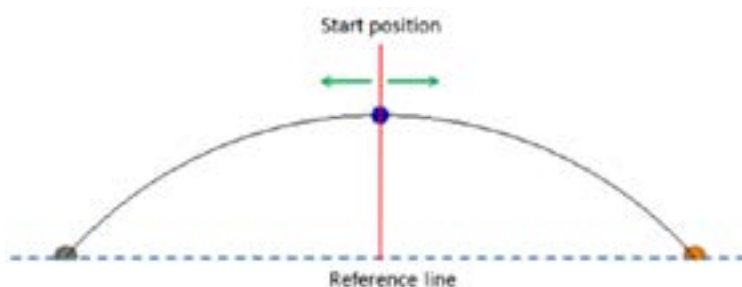
セグメントがフレネル化される前は、スプラインやコニックなどの曲線です。



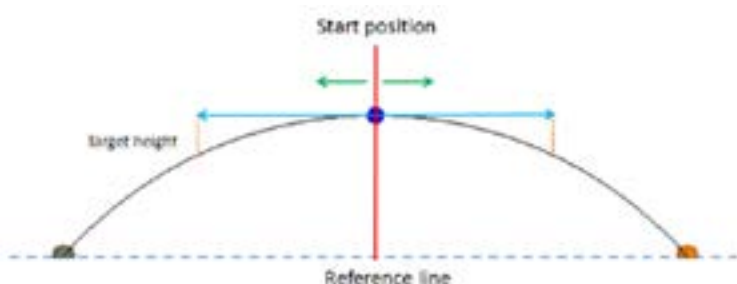
参照線は、2つのセグメント点を結ぶ直線です。



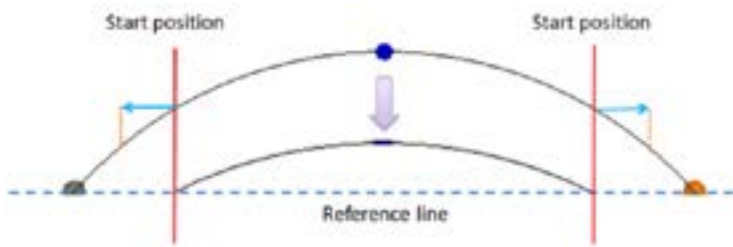
すべてのプリズムが同じ高さになるように、各プリズムの端を探す必要があります。この探索の最初の開始位置は最も高い位置から行われ、この場合、明らかに中心になります。



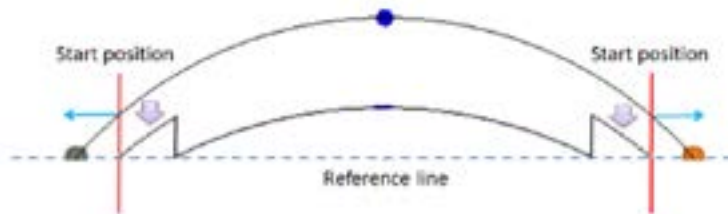
開始位置と探索位置の間のサグ（たるみ）の差が、プロパティエディタで指定されたプリズムの高さと等しくなる適切な位置が見つかったと、このプリズムの端が決定されます。



すると、この領域内の対応するプロファイルは基準線まで引き下げられます。そして、その探索された位置が次の探索の開始位置となります。

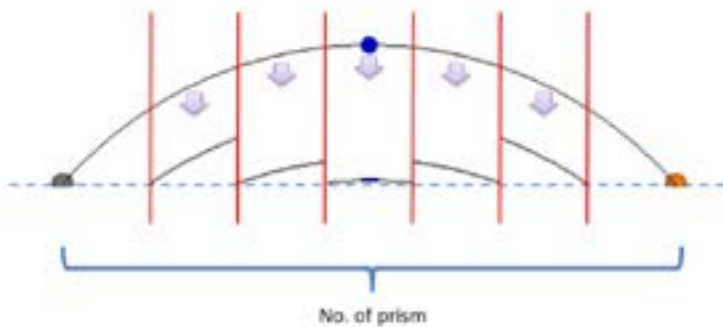


上記のプロセスをすべてのプリズムが見つかるまで繰り返します。



等幅モード：

基準線全体がプロパティエディタで定義されたプリズムの数で分割され、対応するセグメントが地面まで下に移動します。



ベースライン：

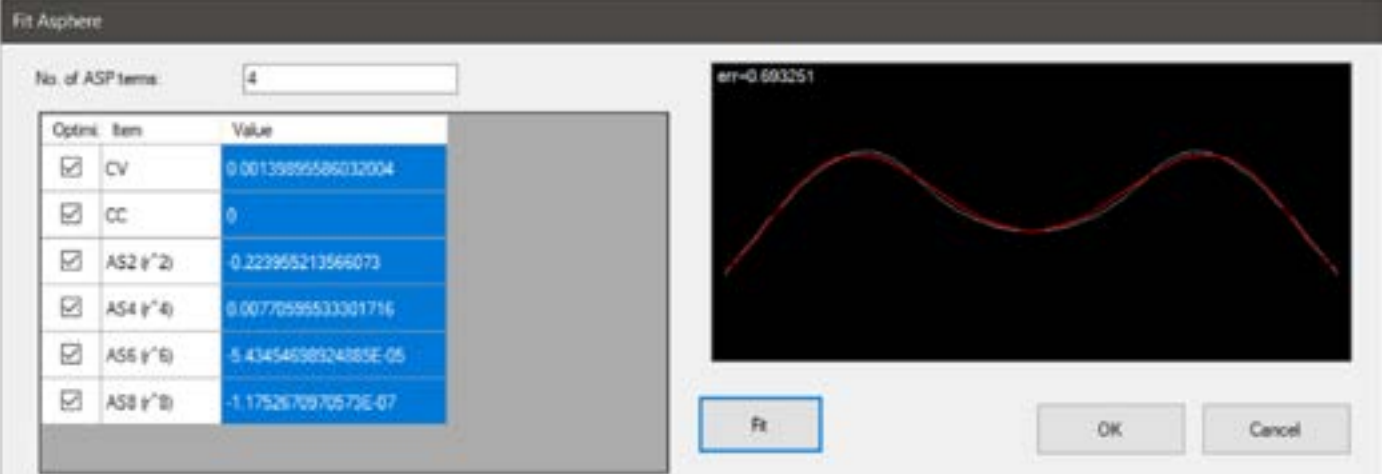
プリズムの形状は、曲線と、2つの端点を結ぶ直線である基準線によって制御されます。しかし、生成されたプリズムは、直線または曲線状のプロファイル（ベースライン）上に配置することができます。

非球面にフィット

説明：

「非球面をフィット」ダイアログは、中央のスプラインセグメントに非球面をフィットさせ、ユーザーは各非球面項の値を取得するためのものです。

「フィット」ボタンをクリックすると、赤い線がフィットされた非球面曲線を表し、非球面パラメータがリスト表示されます。非球面の赤い線と元のスプラインの白い線の差を確認することで、非球面項の数を増やすなどして、スプライン曲線を再度フィットさせる必要があるかどうかを判断します。



The dialog box titled "Fit Asphere" contains a "No. of ASP terms" input field set to 4. Below it is a table with columns "Optim.", "Item", and "Value". The table lists six items, all with checked checkboxes. To the right of the table is a plot area showing a red curve (the fit) and a white curve (the original spline) on a black background. The plot area also displays "err=0.693251". At the bottom are three buttons: "Fit", "OK", and "Cancel".

Optim.	Item	Value
☒	CV	0.00139895586032004
☒	CC	0
☒	AS2 (r^2)	0.223955213566073
☒	AS4 (r^4)	0.00770595533301716
☒	AS6 (r^6)	-5.43454038924085E-05
☒	AS8 (r^8)	-1.1752670970573E-07

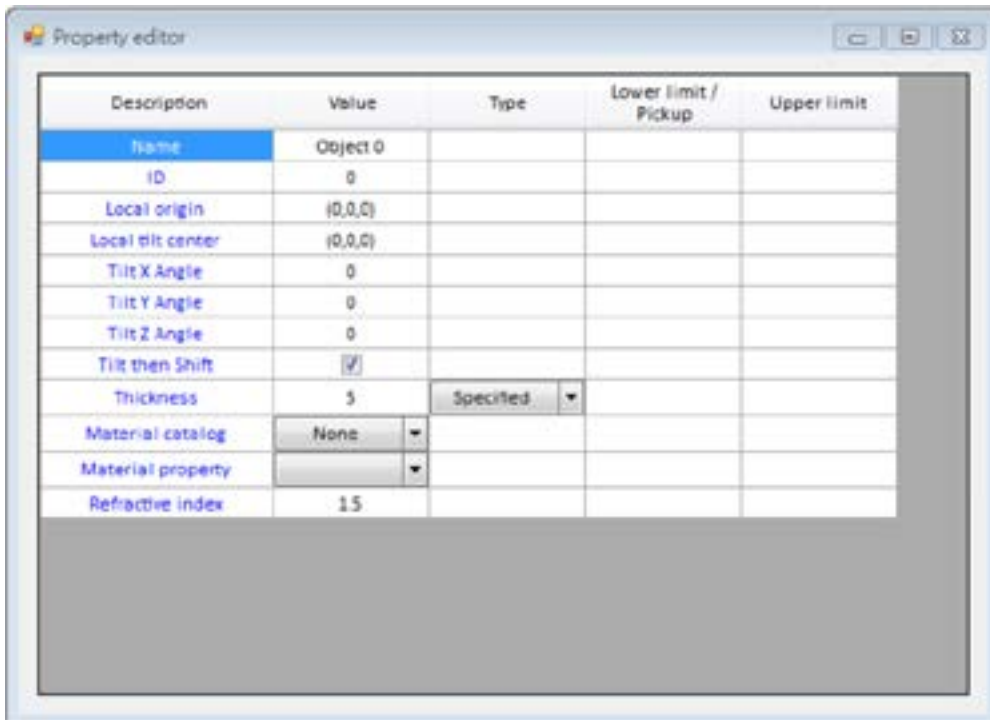
アクション：

No. of ASP terms (非球面項数): 非球面項の数を設定します。新しい値を入力してEnterキーを押すと、データグリッドが更新されます。

Fit (フィット): データグリッド内でチェックされた非球面項を使用して、入力されたスプラインカーブをフィットさせます。

プロパティエディタ

プロパティエディタは、インタラクティブオプティマイザ内のあらゆる種類のコンポーネントのすべてのパラメータと変数を入力および編集するために使用されます。



The screenshot shows a window titled "Property editor" with a table of parameters. The table has five columns: Description, Value, Type, Lower limit / Pickup, and Upper limit. The parameters listed are: name (Object 0), ID (0), Local origin ((0,0,0)), Local tilt center ((0,0,0)), Tilt X Angle (0), Tilt Y Angle (0), Tilt Z Angle (0), Tilt then Shift (checked), Thickness (5, Specified), Material catalog (None), Material property (empty), and Refractive index (1.5).

Description	Value	Type	Lower limit / Pickup	Upper limit
name	Object 0			
ID	0			
Local origin	(0,0,0)			
Local tilt center	(0,0,0)			
Tilt X Angle	0			
Tilt Y Angle	0			
Tilt Z Angle	0			
Tilt then Shift	☒			
Thickness	5	Specified		
Material catalog	None			
Material property				
Refractive index	1.5			

- Property editor items: プロパティエディタ項目
- Pickups: ピックアップ
- Slider Wheel: スライダーホイール
- Function: 関数

プロパティエディタの項目

プロパティウィンドウに表示される項目は、オブジェクトの種類ごとに異なります。各行は、以下の形式で単一のパラメータを記述しています。

Description	Value	Type	Lower limit / Pickup	Upper limit
-------------	-------	------	----------------------	-------------

項目	説明
名称 (Description)	係数の名前
値 (Value)	係数の現在の値
タイプ (Type)	このセルが空の場合、現在の係数が変数またはピックアップとして許可されていないことを意味します。通常、タイプセルは、現在の係数に対して許可されているすべてのオプション（「指定」"Specified"、「相対変数」,"Relative variable"、「絶対変数」,"Absolute variable"、「ピックアップ」"Pick-up".）を含むコンボボックスです。これらのオプションの説明は、変数のセクションにあります。
下限 (Lower limit)	これは、最適化中に現在の係数に許容される最小値を指定します。係数のタイプが「相対変数」"Relative variable" の場合、下限の値は、現在の値から下限の絶対値を引いたものになります。たとえば、現在の値が 6 で、下限が +2 または -2 の場合、許容される最小値は 4 です。「タイプ」"Type" を「ピックアップ」"Pick-up" に設定した場合、このセルにピックアップを制御するための式を入力できます。詳細については、ピックアップのセクションを参照してください。
上限 (Upper limit)	変数に許容される最大値です。上限は下限と同じ方法で計算されます。「タイプ」"Type" が「相対変数」"Relative variable" に設定されている場合、絶対値が使用されます。「ピックアップ」"Pick-up" タイプの場合、セルは空で無視されます。

Pick-up ピックアップ

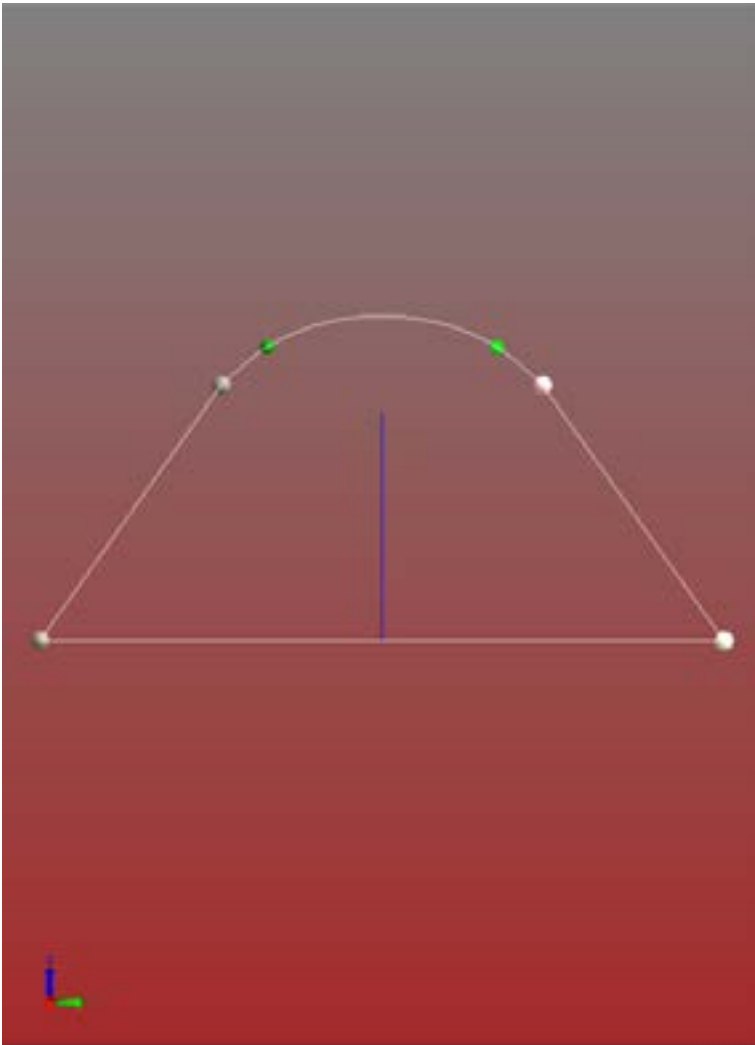
パラメータ値が別のパラメータに依存する必要がある場合は、ピックアップ機能を使用します。他のパラメータから値を取得するには、下限セルにピックアップ方程式を定義します。ピックアップ方程式で値を取得するために使用できるいくつかの特殊関数があります。

Spx(seg_pnt_id, surf_id):	指定されたセグメントポイントの x 値を取得
Spy(seg_pnt_id, surf_id):	指定されたセグメントポイントの y 値を取得
Spz(seg_pnt_id, surf_id):	指定されたセグメントポイントの z 値を取得
Cpx(ctrl_pnt_id, seg_id, surf_id):	指定されたコントロールポイントの x 値を取得
Cpy(ctrl_pnt_id, seg_id, surf_id):	指定されたコントロールポイントの y 値を取得
Cpz(ctrl_pnt_id, seg_id, surf_id):	指定されたコントロールポイントの z 値を取得
Fpx(ctrl_pnt_id, func_id):	指定された関数ポイントの x 値を取得
Fpy(ctrl_pnt_id, func_id):	指定された関数ポイントの y 値を取得
Cplx(cp_id, seg_id, surf_id):	指定されたコントロールポイントのローカル x 値を取得
Cply(cp_id, seg_id, surf_id):	指定されたコントロールポイントのローカル y 値を取得
Cplz(cp_id, seg_id, surf_id):	指定されたコントロールポイントのローカル z 値を取得
seg_prop(prop_nm, seg_id, surf_id):	指定されたサーフェスのセグメントプロパティ値を取得。プロパティの名前は、プロパティエディタに表示されるものと完全に同じである必要があります（例: "Conic constant"、"Curvature"、"Pitch"、"Width" など）。
surf_prop(prop_nm, surf_id):	指定されたサーフェスのサーフェスプロパティ値を取得。プロパティの名前は、プロパティエディタに表示されるものと完全に同一である必要があります、スペースも維持する必要があります。たとえば、双円錐サーフェスのサーフェスプロパティ "X - Curvature" の場合、ピックアップ値を割り当てる際には "X" と "-" の間、および "-" と "Curvature" の間のスペースを維持する必要があります。
obj_prop(prop_nm, obj_id):	指定されたオブジェクトのオブジェクトプロパティ値を取得。プロパティの名前は、プロパティエディタに表示されるものと完全に同一である必要があります、スペースも維持する必要があります。

ピックアップの数式では、いくつかの数学関数もサポートされています。「[数学関数](#)」を参照してください。

ピックアップの使用

プロパティタイプが「ピックアップ」で、ピックアップ方程式のあるセルをクリックすると、サーフェスビューの背景色がクリームゾンに変わり、そのパラメータがピックアップモードであることを示します。



任意の制御点またはセグメント点のパラメータのピックアップオプションは、その点をクリックしてポップアップメニューを開くことで編集できます。



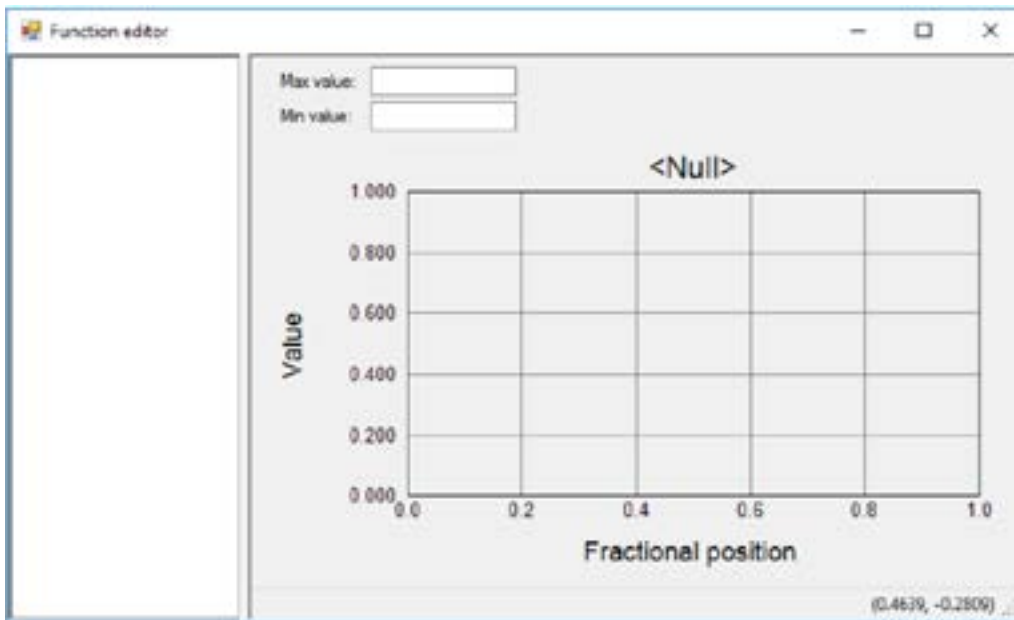
次に、パラメータ選択をクリックして、ピックアップの計算式を編集します。通常のビューに戻るには、Enter キーを押すか、ピックアップセル以外の場所をクリックします。

関数

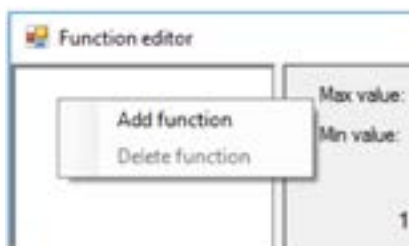
機能は、周期的なパターンに対して連続的に変化する値を定義するために使用されます。例えば、曲線に沿って周期的なパターンのピッチを変化させる必要がある場合、いくつかの制御点を定義することで、曲線上の任意の位置でピッチを異ならせることができます。

新しい機能の作成と修正：

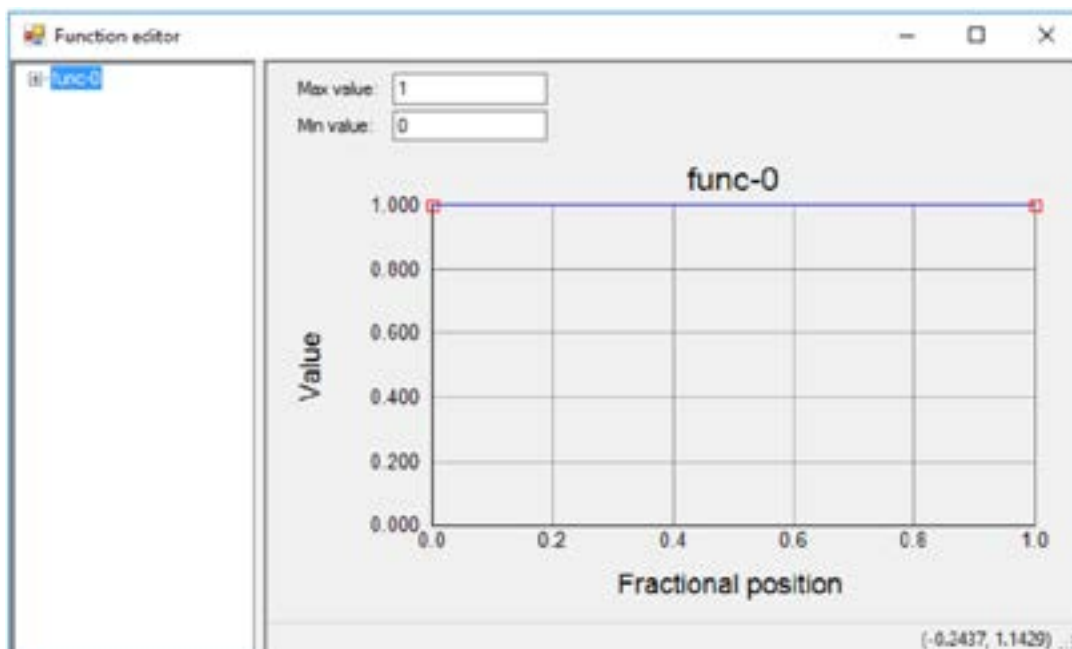
機能を使用するために定義する最初のステップは、新しい機能を関数エディタで作成することです。関数エディタは、ウィンドウ -> 関数エディタ から開くことができます。



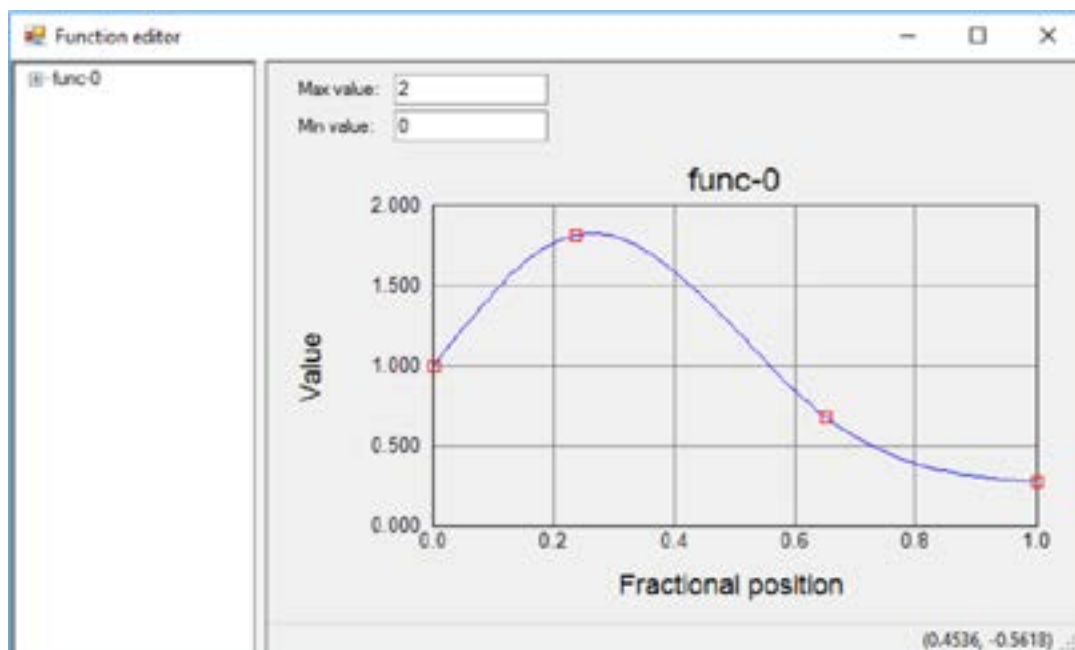
上記の図が示すように、X 軸は適用されたセグメントの相対的な位置を表しているため、0.0 から 1.0 の間になるのは自然なことです。新しい関数を作成するには、左側の空白領域を右クリックし、「関数の追加」を選択して新しい関数を挿入します。



関数名は変更できません。プレフィックス「func-」に続く固有の関数IDとして命名されます。ツリービューで目的の関数をクリックすると、右側のプロットの関数が更新されます。そのデフォルト値は、分数位置 (x) 0.0 から分数位置 (x) 1.0 まで、一定値 1.0 です。



制御点をドラッグしたり、プロット上で右クリックしてさらに制御点を挿入したりすることで、関数曲線をさらに操作して、期待する形状に合わせることができます。



制御点を変数として設定とピックアップ：

関数の選択された制御点は、プロパティエディタウィンドウで編集可能です。通常の制御点がセグメントのスケッチであると同様に、関数点も変数として扱ったり、他の点から値をピックアップしたりすることができます。ただし、変数として扱う場合、許容されるタイプは相対変数のみです。

変数として設定したり、他の制御点から値をピックアップしたりする操作は、通常の制御点と同じです。

周期パターンを持つ関数の使用：

関数が作成されると、周期セグメントや、高度なスイープオブジェクトのパススケールパラメータとして使用できます。周期セグメントの場合、円形および三角形フィーチャのピッチ、半径、高さは関数値にすることができ、三円形フィーチャのピッチ、中心半径、周辺半径も同様です。

係数を関数値として設定するには、まずパラメータのタイプを「Func」として選択する必要があります。

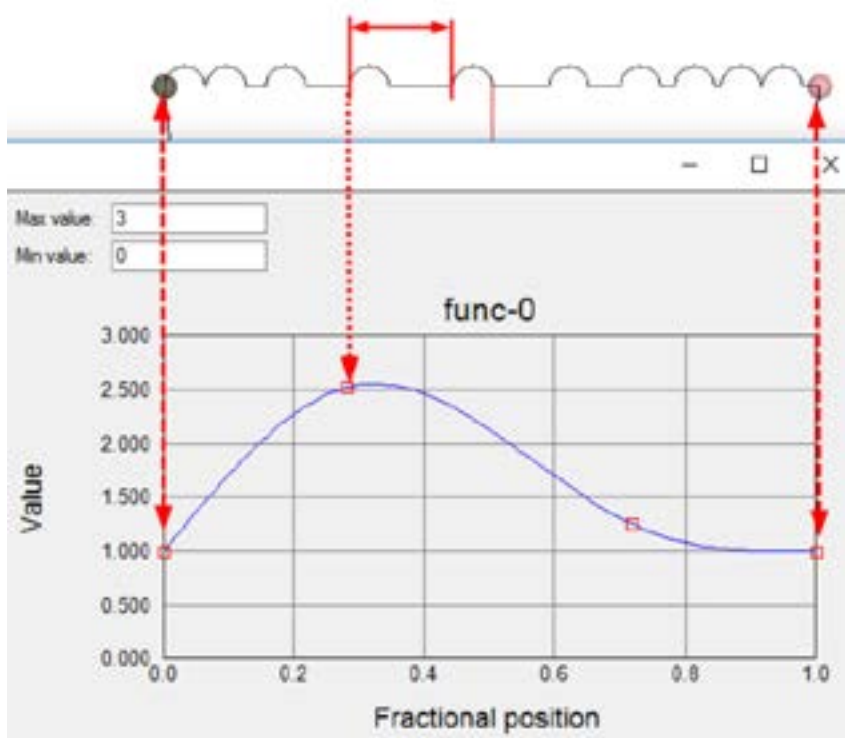
Property editor

Description	Value	Type	Lower limit / Pickup	Upper limit
ID	1			
Segment type	Periodic			
Geometry type	Circle			
Pitch	2	Specified		
Radius	0.5	Specified		
Height	0.5	Specified		
Reflective?	☐			
Absorptive?	☐			
Center feature option	Space			

次に、値セル内のテキストを選択したパラメータに紐付けたい関数名に変更します。この例では、「func-0」が先ほど作成した関数名です。

Property editor

Description	Value	Type	Lower limit / Pickup	Upper limit
ID	1			
Segment type	Periodic			
Geometry type	Circle			
Pitch	func-0	Func		
Radius	0.5	Specified		
Height	0.5	Specified		
Reflective?	☐			
Absorptive?	☐			
Center feature option	None			
Shift	0	Specified		



スライダーホイール

このスライダーホイールは、ユーザーがプロパティ項目のデジタル値を調整するのに役立ちます。スライダーホイールを起動するには、右クリックして「スライダーホイールとして追加」を選択し、スライダーホイールウィンドウをポップアップ表示します。スライダーホイールウィンドウでは、テキストボックスまたはスライダーホイールを介して値を入力できます。また、スライダーホイールの刻み値を調整するための垂直スクロールバーがあります。



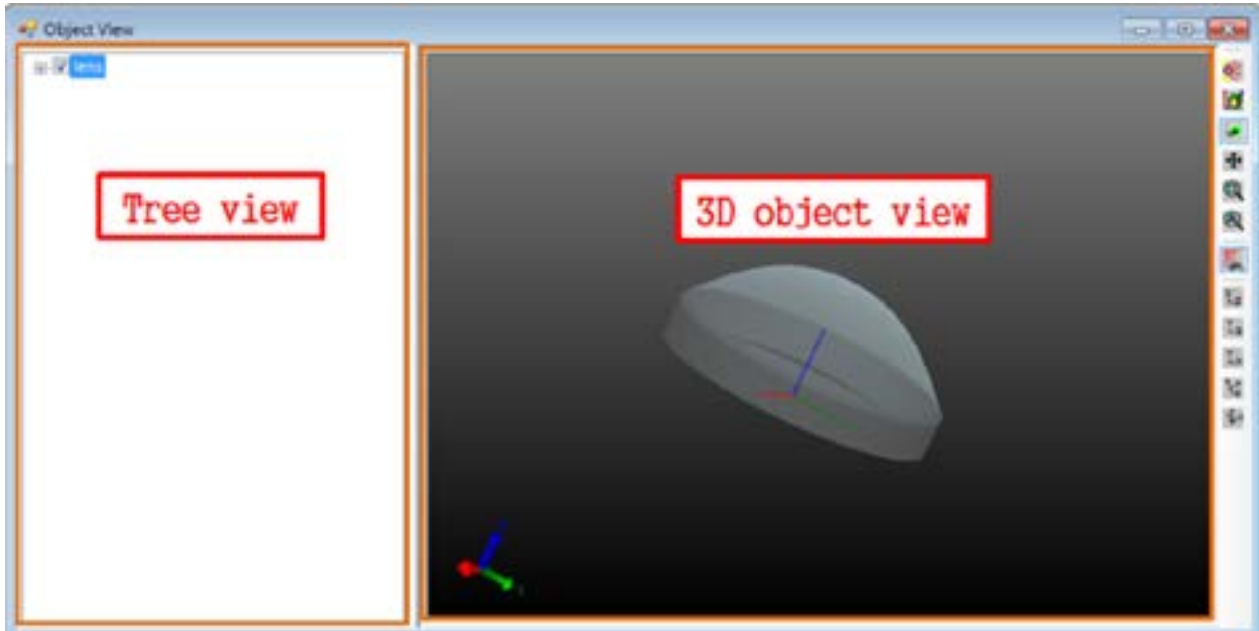
The screenshot shows a window titled "Sliders Wheel" with a table of four rows. Each row represents a property of a surface, with columns for the surface ID, segment point ID, position type, current value, and a scale factor. The values are displayed in a text box with a slider wheel icon to its right.

Surface (ID:0)	Seg Pnt (ID:2)	Position-Y	8.385	0.02
Surface (ID:0)	Seg Pnt (ID:2)	Position-Z	5	0.01
Surface (ID:0)	Seg Pnt (ID:3)	Position-Y	7.42234182	0.01
Surface (ID:0)	Seg Pnt (ID:3)	Position-Z	0.01434454	0.001

オブジェクトビュー

作成したオブジェクトはすべて、移動およびサイズ変更可能なウィンドウであるオブジェクトビューに、3D レンダリングされた状態で表示されます。

サーフェスビューと同様に、オブジェクトビューはツリービューと 3D オブジェクトビューの 2 つの部分に分かれています。



その他：

- 編集機能
- ツールバーコマンド

編集機能

機能：

a. 新しいオブジェクトを追加するには：

ツリー表示を右クリックし、「オブジェクトを追加」を選択します。既存のサーフェスから新しいオブジェクトを作成するのに役立つ新しいダイアログが表示されます。オブジェクトの種類のセクションで説明されているように、5 種類のオブジェクトがあります。

b. 1 つまたは複数のオブジェクトを削除するには：

削除するオブジェクトを選択し、ツリー表示を右クリックして「オブジェクトを削除」を選択します。

c. オブジェクトを修正するには：

既存のオブジェクトの係数を修正するには、右クリックして「オブジェクトを修正」を選択するか、プロパティエディタウィンドウで係数を編集します。オブジェクトの種類を変更したり、既存のオブジェクトの対応 / リンクするサーフェスを修正したりする場合は、「オブジェクトを修正」の選択を使用する必要があります。利用可能なオブジェクトの種類とその説明は、オブジェクトの種類のセクションにあります。

d. オブジェクトをコピー / ペーストするには：

ツリー表示を使用してコピーするオブジェクトを選択し、右クリックして「コピー」（または Ctrl-C）を選択します。ペーストするには、右クリックして「ペースト」（または Ctrl-V）を選択します。コピーされたオブジェクトとペーストされたオブジェクトの唯一の違いは、ペーストされたオブジェクトのオブジェクト ID が増加することです。

e. TracePro へ送信：

最適化のためにオブジェクトを TracePro に送信するには、ツリー表示でオブジェクトを選択し、右クリックして「TracePro へエクスポート」を選択します。

f. ソースを定義する：

3D ビューで直接ソースを 1 つ作成できます。まず、ツールバーのソースアイコン () を押して、3D ビューをソースモードに切り替えます。オブジェクトビューウィンドウがソースモードになったら、右クリックして「ソースを定義」を選択し、新しいソースを作成します。既存のソースがある場合は、左クリックしてプロパティエディタウィンドウでパラメータを表示し、すべてのパラメータを編集できます。ソースの詳細については、「ソースの定義」(Define Source) を参照してください。

g. オブジェクトを移動する：

まず、移動オブジェクト () アイコンをクリックし、マウスを使用して 3D ビューでオブジェクトをドラッグします。

h. ビュー調整：

3D ビューを右クリックして、「照明」と「アルファブレンディング」の選択肢を見つけます。その他の表示機能には以下が含まれます。

1. マウスホイールを使用したズームイン / ズームアウト
2. 3D ビュー内をダブルクリックしてビューを原点に中心を合わせる

光線表示の設計意図、挙動、および制限事項：

インタラクティブオブティマイザを使用すると、サーフェスリストウィンドウとオブジェクトビューウィンドウでソースを定義し、光線をトレースできます。オブティマイザの光線には波長の値はありません。オブティマイザは、屈折率の計算にオブティマイザの設定を使用します。屈折率は、サーフェスリストと

オブジェクトビューのプロパティエディタで設定できます。それらの値が同じである必要はありません。ユーザーは、サーフェスを反射性または吸収性として設定するオプションもあります。インタラクティブ オプティマイザは、オプティマイザでの光線追跡に、オプティマイザで適用されたサーフェスプロパティまたは材料プロパティを使用しません。光線は鏡面反射、鏡面透過、または吸収されます。散乱はモデル化されません。これらのパラメータはすべて TracePro でモデル化されます。

説明：	係数の名称
値：	係数の現在の値
タイプ：	このセルが空の場合、現在の係数が変数またはピックアップとして使用できないことを意味します。通常、タイプセルは、現在の係数に対して許可されているすべてのオプション（「指定」"Specified"、"Relative variable"、"Absolute variable"、"Pick-up".）を含むコンボボックスです。これらのオプションの説明は、変数のセクションにあります。
下限：	これは、最適化中に現在の係数に対して許容される最小値を指定します。係数のタイプが「相対変数」"Relative variable" の場合、下限の値は現在の値から下限の絶対値を引いたものになります。例えば、現在の値が 6 で下限が +2 または -2 の場合、許容される最小値は 4 です。「タイプ」Type を「ピックアップ」"Pick-up" に設定した場合、このセルにピックアップを制御するための数式を入力できます。詳細については、ピックアップのセクションを参照してください。
上限：	変数に対して許容される最大値です。上限の計算方法は下限と同じです。「タイプ」Type が「相対変数」"Relative variable" に設定されている場合、絶対値が使用されます。「ピックアップ」タイプの場合、このセルは空で無視されます。

ツールバーコマンド

ツールバーには、いくつかの定義済みのビューも用意されています。

XZ ビュー：	正の X 軸を上、正の Z 軸を右に表示します。
YZ ビュー：	正の Y 軸を上、正の Z 軸を右に表示します。
YX ビュー：	正の Y 軸を上、正の X 軸を右に表示します。
Isometric View1 アイソメトリックビュー 1:	正の Y 軸を上、正の X 軸を奥へ、正の Z 軸を手前に表示します。
Isometric View2 アイソメトリックビュー 2:	正の Y 軸を上、正の X 軸と正の Z 軸を奥へ表示します。

アプリケーションウィンドウの右側にあるツールバーには、オブジェクトを編集するための最も一般的なコマンドが用意されています。

ソース定義： 	光線追跡のためのソースを定義 / 修正できる光線追跡モードにします。ソースの作成プロセスは、 ソース定義 のページで説明されています。
オブジェクト移動： 	マウスでオブジェクトをドラッグして移動します。
オービットビュー： 	マウスを使用してビューを回転させます。
パンカーソル： 	マウスを使用してビューを平行移動します。
ウィンドウズーム： 	マウスをドラッグして領域を定義し、そのウィンドウ領域にズームします。
全画面ズーム： 	すべてのオブジェクトを余白を含めて表示するようにズームし、再中心に配置します。
光線表示 / 非表示： 	チェックが入っている場合、定義されたソースからの光線がモデルウィンドウに表示されます。

オブジェクトのタイプ

TracePro では、設計と光線追跡にソリッドオブジェクトを使用します。しかし、光学設計では、サーフェスが光学システムを定義するために使用されます。インタラクティブオブティマイザでは、ハイブリッドなアプローチを採用しています。サーフェスビューアを使用すると、サーフェスまたはプロファイルを簡単に作成し、それらを組み合わせてソリッドオブジェクトを作成できます。インタラクティブオブティマイザを使用すると、レンズ、放射対称オブジェクト、押し出しオブジェクト、スイープオブジェクト、二軸オブジェクトなど、いくつかの種類のソリッドジオメトリを構築できます。すべてのオブジェクトタイプには、オブジェクトのプロパティを構成するいくつかの共通パラメータがあります。

ID:

オブジェクトの一意の番号。ID 番号は、オブジェクトを作成すると自動的に割り当てられ、編集することはできません。

名前:

識別しやすいように、任意のオブジェクトの名前を変更できます。これは、アフタースキームプログラミングで特に役立ちます。

変換:

変換行列は、オブジェクトの位置と向きを定めるために使用される座標変換行列です。便宜上、インタラクティブオブティマイザでは、完全な行列を入力する代わりに、オイラー角の回転と並進を設定するように求められます。オブジェクトはリンクされたサーフェスに基づいて作成されますが、その実際の位置と向きは変換行列によって決定されます。位置は、ローカル座標系のワールド座標系に対するオフセット、つまり、ローカル原点がワールド座標系内のどこに位置するかです。向きは、それぞれ x、y、z 軸を中心とした連続的な回転によって決定されます。オブジェクトのローカル原点とは別に、回転中心に個別の原点を割り当てることもできます。「傾斜してからシフト」オプションを選択すると、回転と並進の順序を逆にすることもできます。

詳細:

Object types: Lenses (レンズ)

Object types: Extruded object (押し出しオブジェクト)

Object types: Radially symmetric object (放射対称オブジェクト)

Object types: Sweep object (スイープオブジェクト)

Object types: Biaxial object (二軸オブジェクト)

Object types: Reflector (反射体)

Object types: TIR lens (TIR レンズ)

Object types: Freeform lens (フリーフォームレンズ)

Object types: Reptile object (レプタイルオブジェクト)

オブジェクトタイプ：レンズ

レンズは通常、両面を持つオブジェクトです。

リンクされたサーフェス数：2

リンクされたサーフェスタイプ：

平面、B- スプライン、双円錐サーフェスを含む、両方のタイプ A サーフェス

ねじれたオブジェクトを構築するための、両方のタイプ B サーフェス

厚さ：レンズの厚さは、2つの基準平面間の距離です。双円錐 - 双円錐レンズの場合、厚さは頂点から頂点までの長さであり、B- スプラインサーフェスの場合は、レンズの実際の厚さは、平面間の距離とサーフェス中心における B- スプラインの高さの合計です。

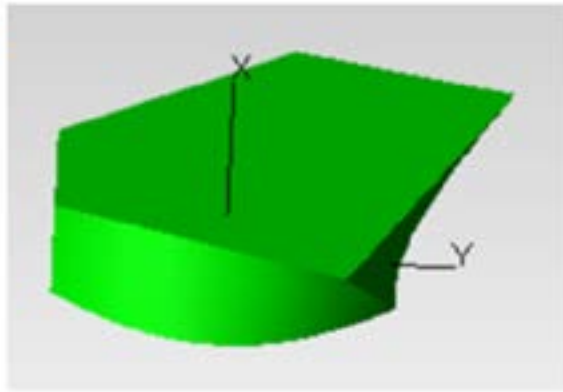
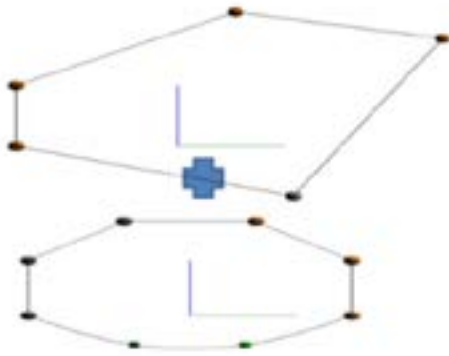
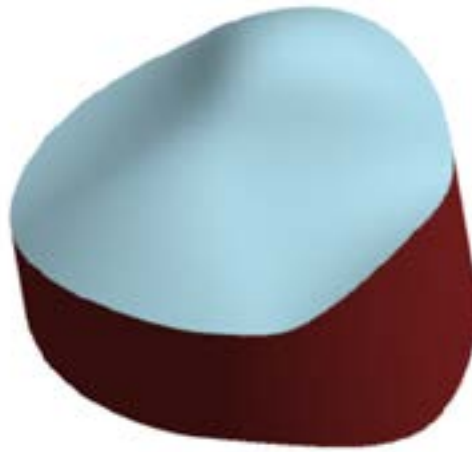
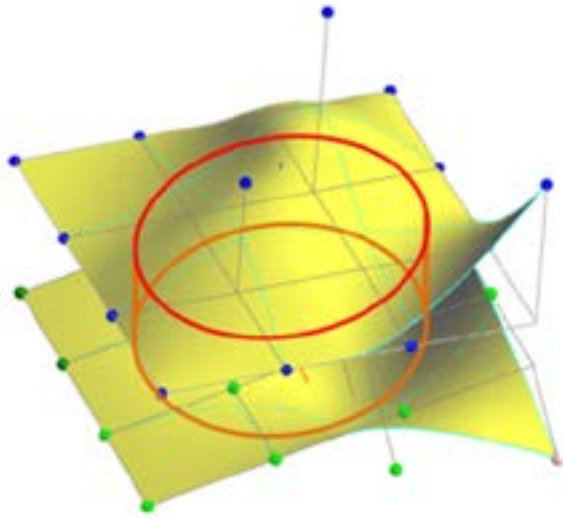
タイプ B サーフェスの場合、2つの基準サーフェスは厚さの半分の距離に配置され、インタラクティブオブティマイザは上下のサーフェスの頂点間の誤差を最小化し、結果として得られるジオメトリを（ねじれた）オブジェクトとして作成します。

アパーチャ形状：円形または長方形（選択されたリンクされたサーフェスタイプがタイプ A の場合のみ利用可能）

半径：アパーチャ半径（選択されたリンクされたサーフェスタイプがタイプ A の場合のみ利用可能）

The screenshot shows a software dialog box titled "Add a new object". It contains the following fields and controls:

- Name:** A text field containing "Object 3".
- Type:** A dropdown menu set to "Lens".
- Transform:**
 - Location:** A text field containing "(0,0,0)".
 - Local Tilt center:** A text field containing "(0,0,0)".
 - Tilt then Shift:** A checked checkbox.
 - X angle:** A text field containing "0" with "deg" next to it.
 - Y angle:** A text field containing "0" with "deg" next to it.
 - Z angle:** A text field containing "0" with "deg" next to it.
- Linked surface #1:** A dropdown menu set to "Surface 4(ID:4)".
- Linked surface #2:** A dropdown menu set to "Surface 5(ID:5)".
- Reverse:** Two unchecked checkboxes, one next to each linked surface dropdown.
- Aperture shape:** A dropdown menu set to "Circular".
- Thickness:** A text field containing "3".
- Radius:** A text field containing "4".
- Buttons:** "Apply" and "Cancel" buttons at the bottom right.



オブジェクトタイプ：押し出しオブジェクト

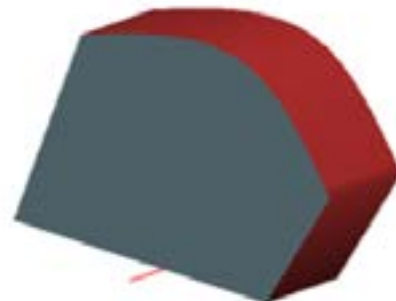
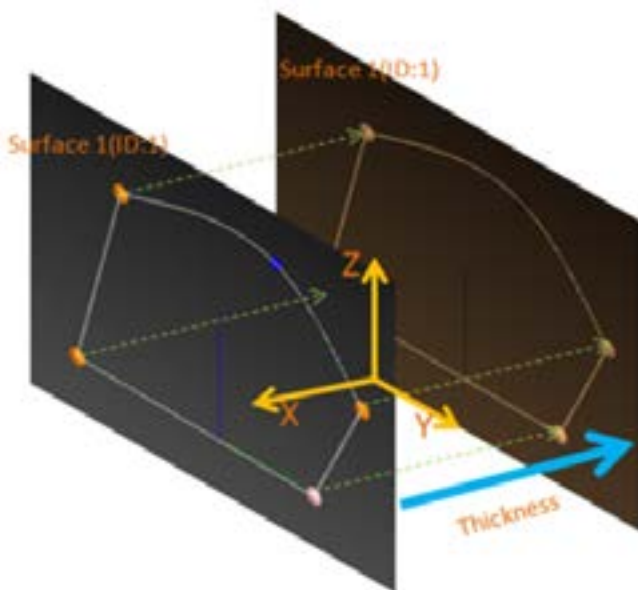
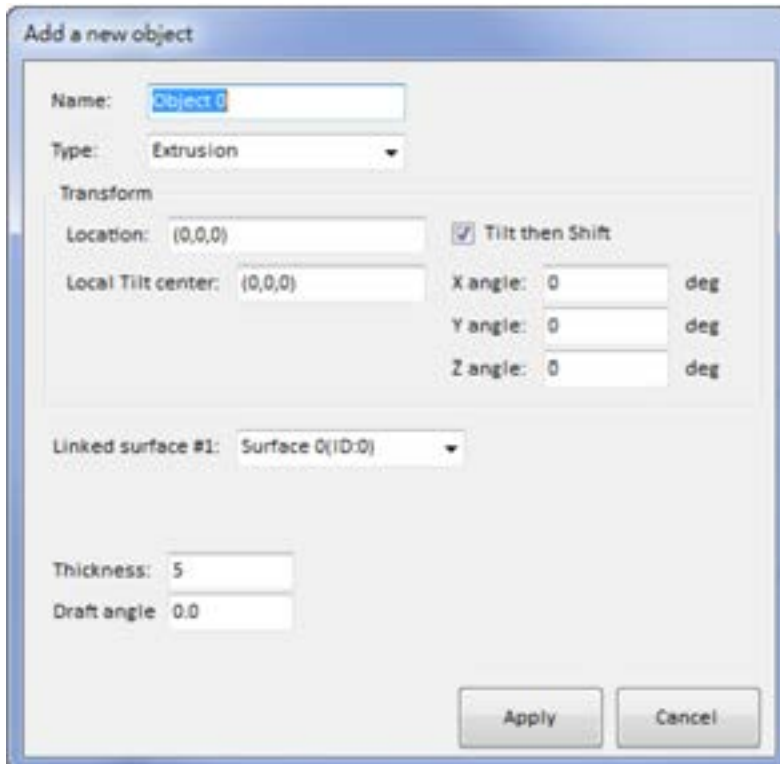
押し出しオブジェクトは、平面的な輪郭を指定された距離に沿って押し出すことによって作成されるソリッド（立体）です。

リンクされたサーフェス数：1

リンクされたサーフェスタイプ：タイプ A サーフェス、タイプ B サーフェス、タイプ D サーフェス

厚さ：押し出しの長さ

抜き勾配：押し出し中に抜き勾配を指定します。



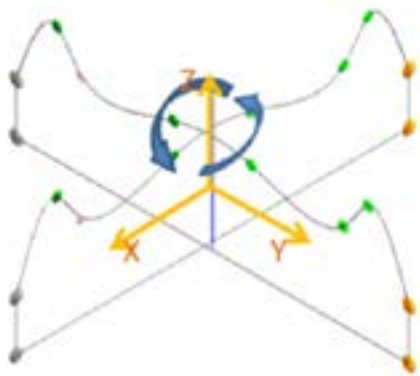
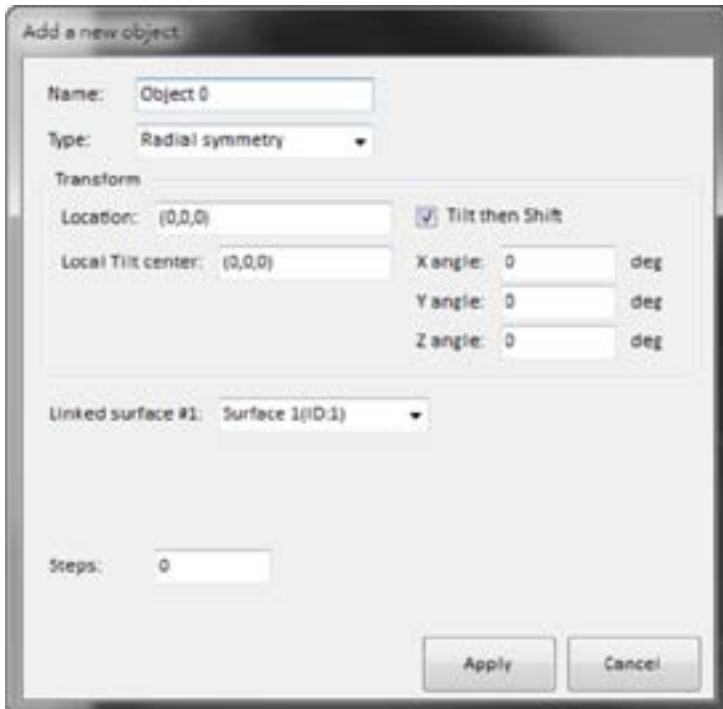
オブジェクトタイプ：放射対称オブジェクト

対称プロファイル面は、放射状のプロファイルを中央軸を中心に回転させることによって作成されます。これは、回転対称なレンズを作成するのに適しています。

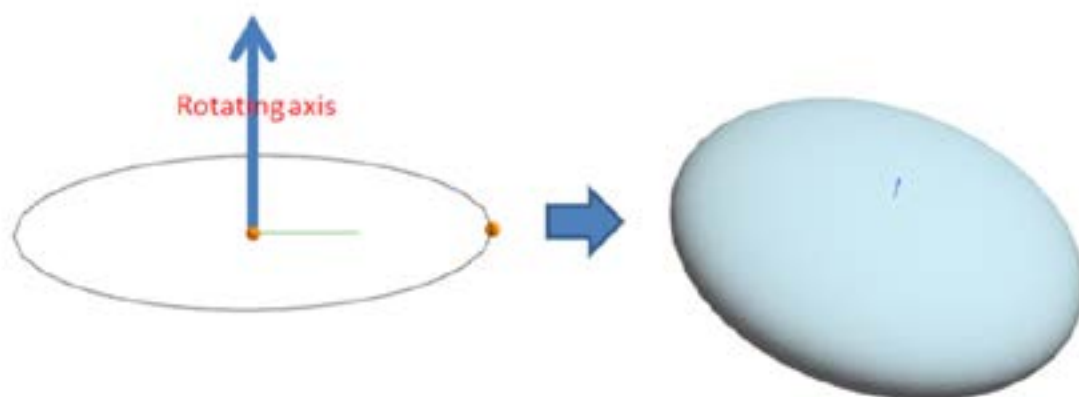
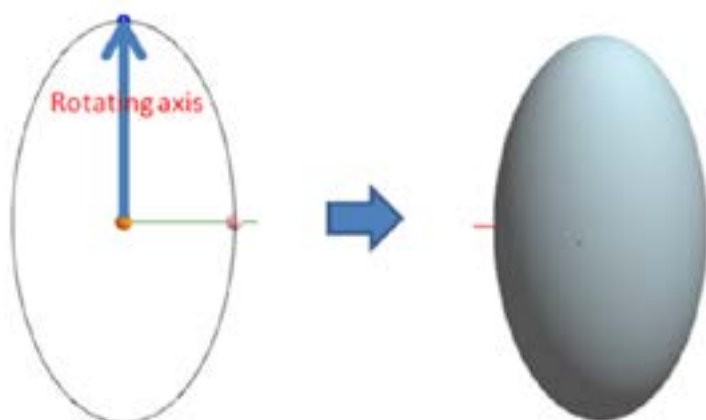
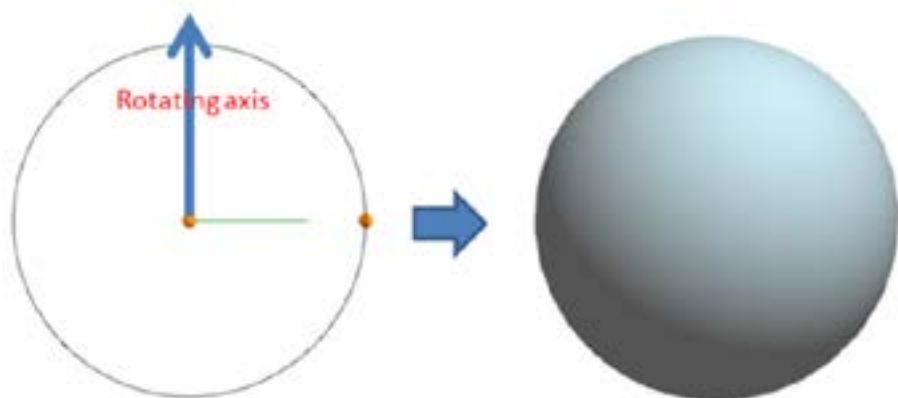
リンクされたサーフェス数：1

リンクされたサーフェスタイプ：対称プロファイルと楕円プロファイル

ステップ数：回転軸を中心に回転させて多角形形状のオブジェクトを作成する際のステップ数を定義します。ステップ数がゼロの場合、連続的な回転が適用されます。



楕円形の断面形状から放射状に対称なオブジェクトを作成することで、球体または回転楕円体（扁球または長球）を構築できます。

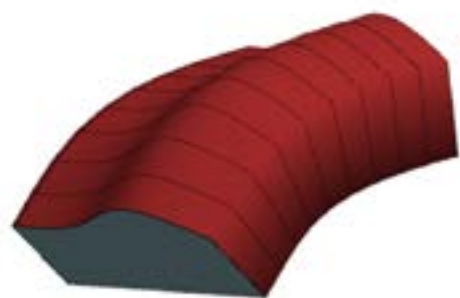
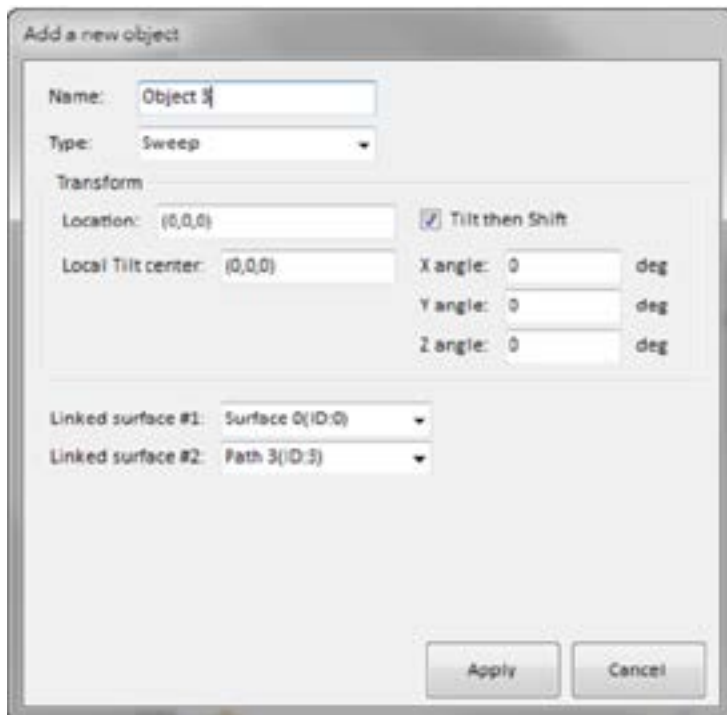


オブジェクトタイプ：スイープオブジェクト

「スイープオブジェクトは押し出しオブジェクトに似ていますが、スイープオブジェクトは平面をあらかじめ定義されたパスに沿って掃引（スイープ）することでソリッドを作成します。スイープオブジェクトの作成に使用するプロファイル面は、タイプ B の面である必要があります。

リンクされた面の数：2

リンクされた面のタイプ：面 #1 はタイプ B の面である必要があります、面 #2 はタイプ C の面（2D パスまたは 3D パス）である必要があります。

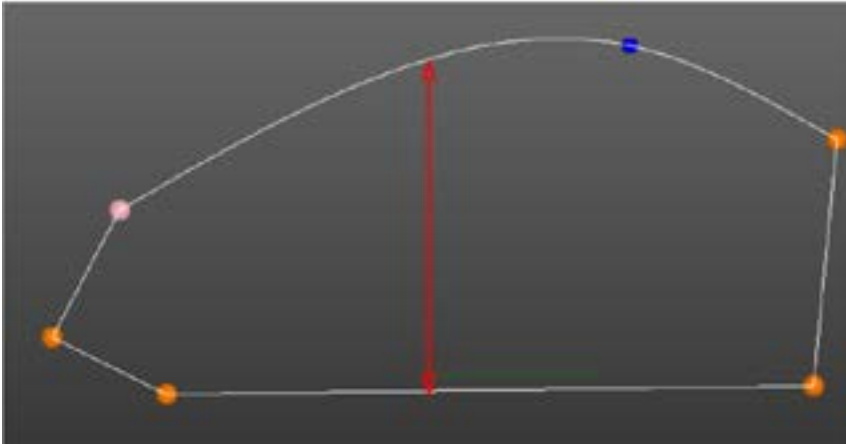


オブジェクトタイプ：バイアキシャルオブジェクト

バイアキシャルオブジェクトは、互いに垂直な2つのプロファイルから作られます。その表面は、方位角方向に沿ってこれらのプロファイル間を線形に補間することで生成されます。ただし、補間された表面の形状は予測できない場合があります。たとえば、バイアキシャルオブジェクトに対して2つの円錐プロファイルを選択しても、双円錐面にはなりません。

対称または非対称の2つのプロファイルをバイアキシャルオブジェクトとして組み合わせるには、以下の重要な点に注意を払う必要があります。

結果として得られるバイアキシャルオブジェクトの厚さは、最初のプロファイルの中央の厚さによって決まります。「中央」という用語は、下の赤い矢印で示されているように、 $y=0.0$ の位置を指しています。

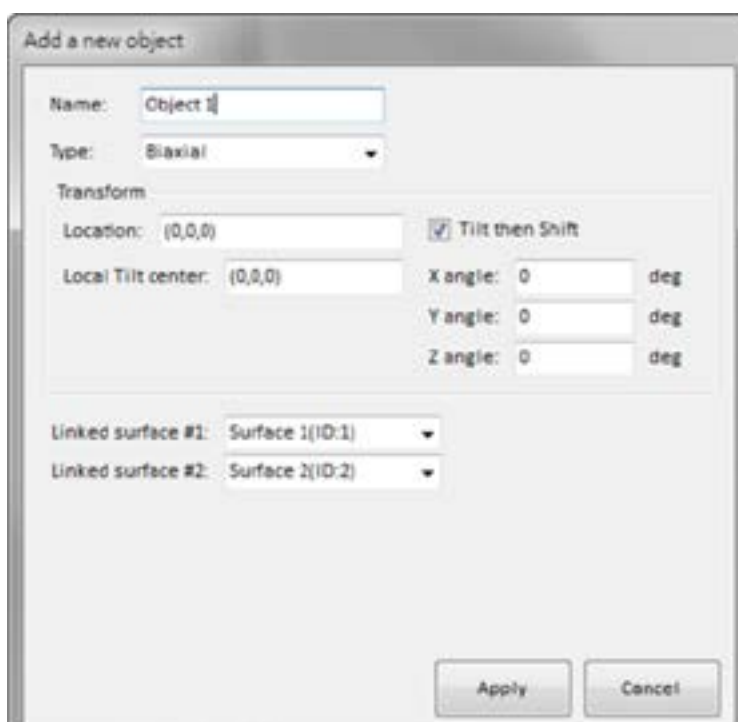


2 番目のプロファイルは、2つのプロファイルが互いに接合するように、スケール調整とシフト処理が行われます。言い換えれば、2つのプロファイルを指定する順序によって、異なる二軸オブジェクトが生成されます。また、これは、選択された2つのプロファイルが正と負の両方の側面を持つ必要があるという制約を引き起こします。つまり、プロファイルは垂直軸（z 軸）を横切る必要があります。

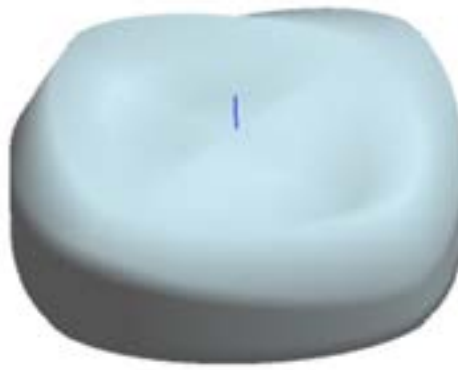
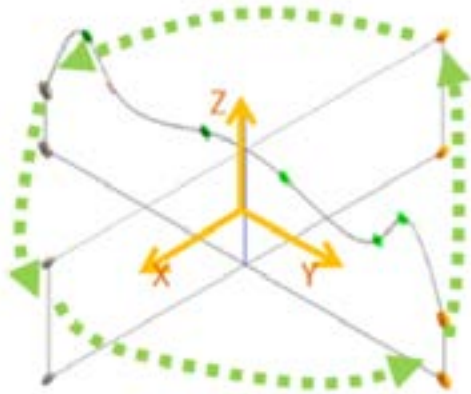
2.

リンクされたサーフェス数:2

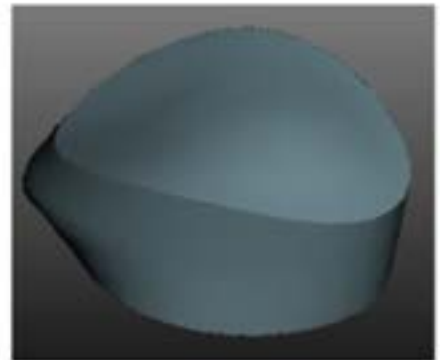
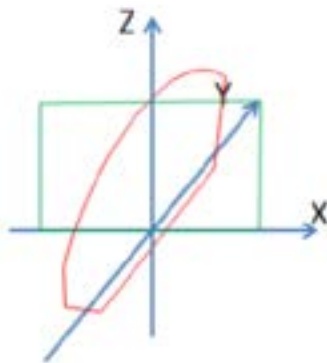
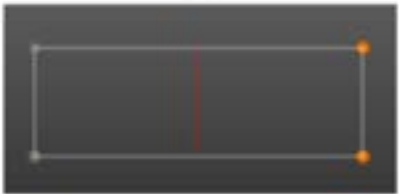
リンクされたサーフェスタイプ: 対称 / 非対称プロファイルサーフェス



二つの対称な断面形状を持つ二軸性の物体（光学素子）



非対称プロファイルの場合



オブジェクトタイプ：反射板

オブジェクト「反射板」は、反射体の反射面をファセット化（細かく分割）するニーズを満たすように設計されています。

リンクされたサーフェス数：1

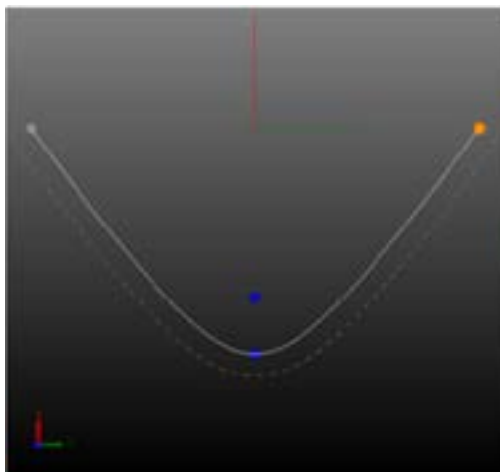
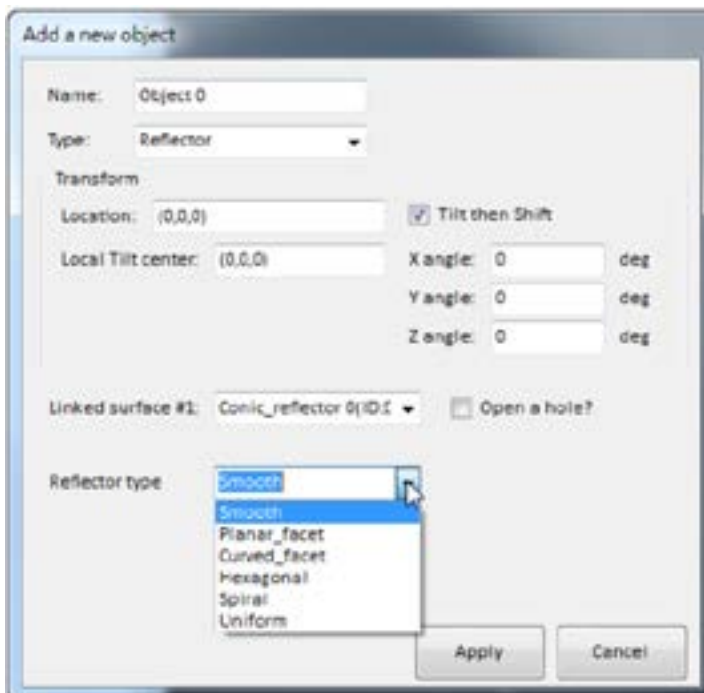
リンクされたサーフェスタイプ：スプライン反射板、円錐反射板

反射板タイプ：スムーズ、平面ファセット、曲面ファセット、六角形、螺旋、均一

ステップ：セグメントの数

リング：リングの数

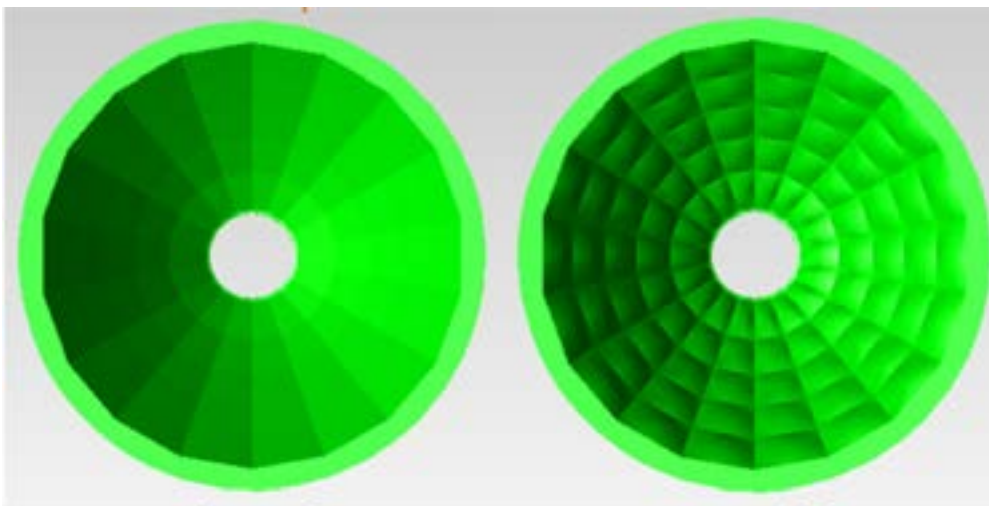
穴を開けますか？：生成された反射板の底面に穴を開ける場合にチェックします。



Reflector profile



Smooth

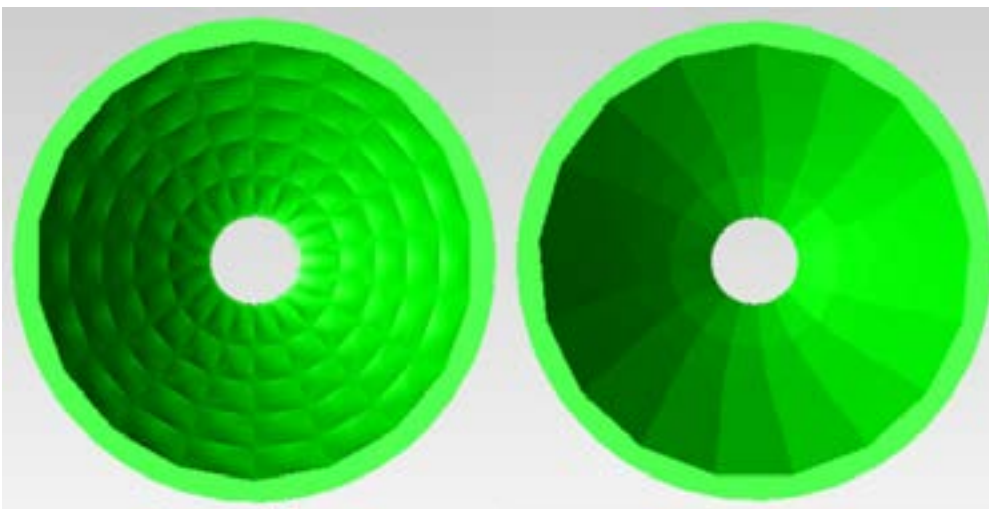


Planar facet

Steps:18
Rings: 6
Hole: Yes

Curved facet

Steps:18
Rings: 6
Hole: Yes
Facet height: 0.3

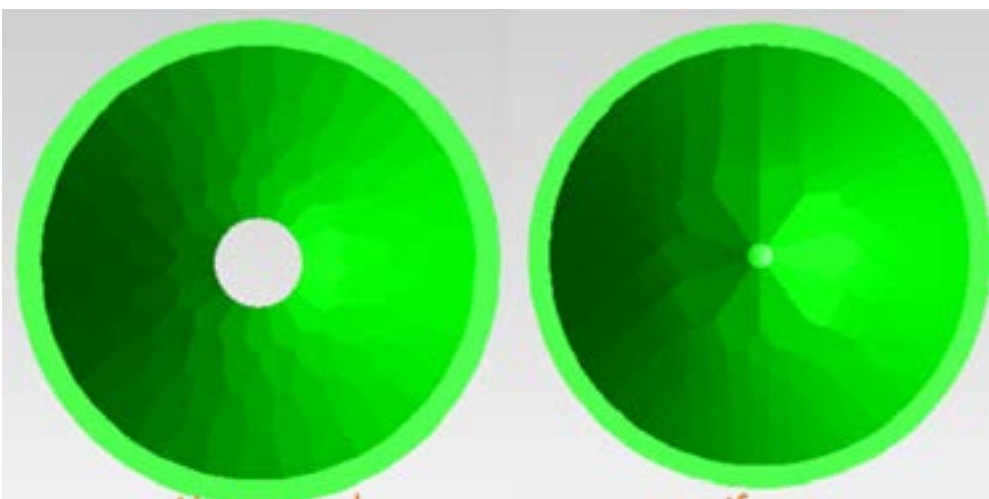


Curved facet

Steps:18
Rings: 6
Hole: Yes
Facet height: -0.3

Spiral

Steps:18
Rings: 6
Hole: Yes
Spiral angle: 10.0



Hexagonal

Steps:18
Rings: 6
Hole: Yes

uniform

Steps:36
Rings: 12
Hole: No

オブジェクトタイプ：TIR レンズ

オブジェクト - TIR レンズは、TIR レンズプロファイルを持つ TIR（全反射）レンズを作成するために設計されています。

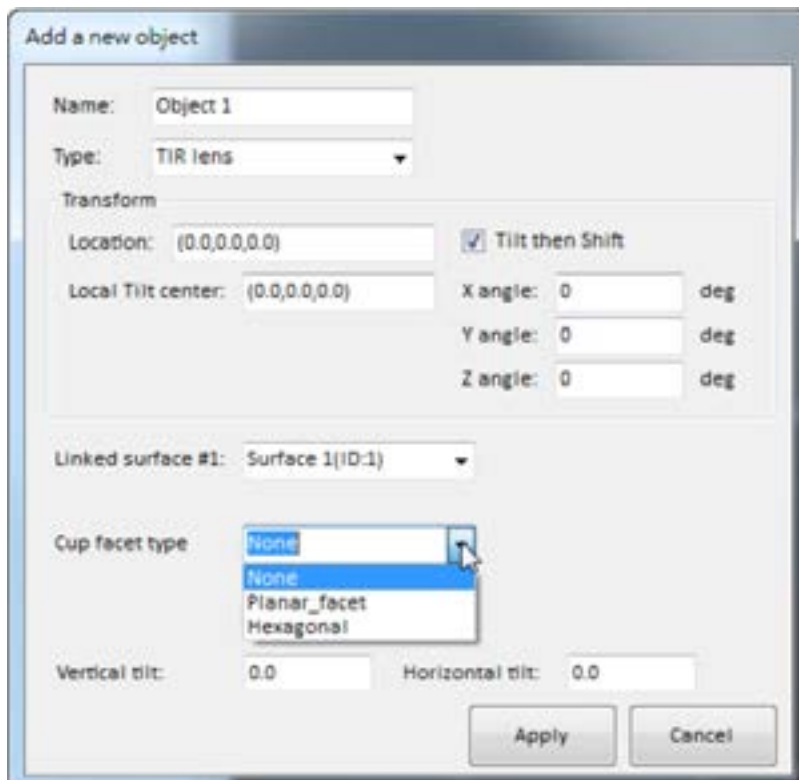
リンクされたサーフェス数：1

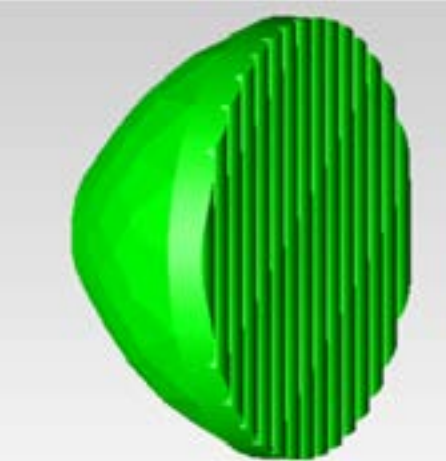
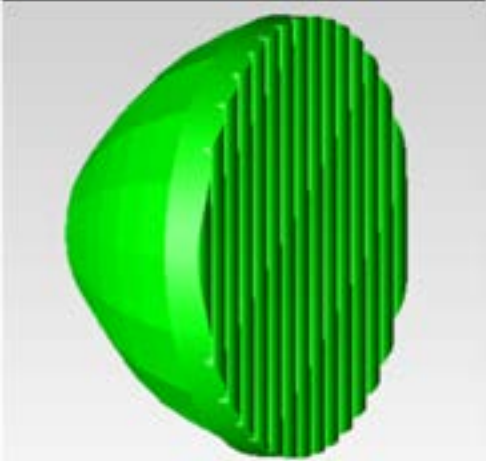
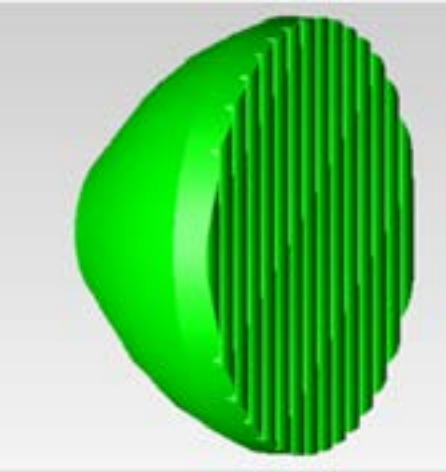
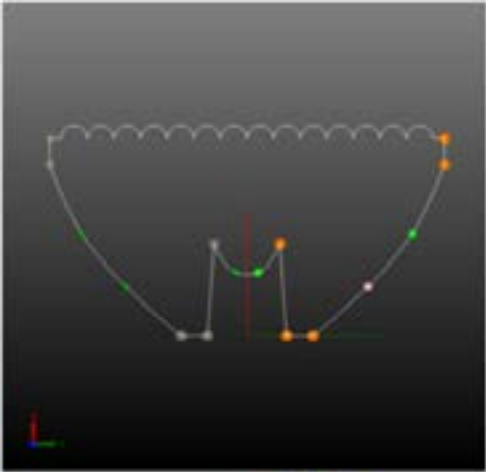
リンクされたサーフェスタイプ：TIR レンズプロファイル

カップファセットタイプ：なし、平面ファセット、六角形

垂直方向の傾斜：出射面の垂直方向の傾斜角度

水平方向の傾斜：出射面の水平方向の傾斜角度





オブジェクトタイプ：自由曲面レンズ

オブジェクト - 自由曲面レンズは、自由曲面を持つレンズを作成するために設計されています。

リンクされたサーフェス数：1 または 2。単一の自由曲面を使用すると凸平レンズを構築し、2 つ使用すると凸凹レンズを作成します。

リンクされたサーフェスタイプ：自由曲面

Add a new object

Name: Object 0

Type: Freeform lens

Transform

Location: (0,0,0,0,0,0) ☒ Tilt then Shift

Local Tilt center: (0,0,0,0,0,0)

X angle: 0 deg

Y angle: 0 deg

Z angle: 0 deg

Linked surface #1: Surface 0(ID:0)

Linked surface #2: Surface 1(ID:1)

Apply Cancel

オブジェクトタイプ：アドバンスドスイープオブジェクト

アドバンスドスイープオブジェクトは、スイープオブジェクトを進化させたもので、ユーザーが複数のプロファイル（Type B サーフェス）を組み合わせることで複雑なパイプ部品を構築できます。同時に、パス（2D または 3D）をアドバンスドスイープオブジェクトのスイープ軌道として使用することもできます。

リンクされたサーフェス数：2 以上

リンクされたサーフェスタイプ：プロファイルとして Type B サーフェス、パスとして Type C サーフェス。

プロファイルは、結果として得られるオブジェクトを構成する断面形状を定義し、パスはプロファイルの位置と方向を定めるための経路を提供します。

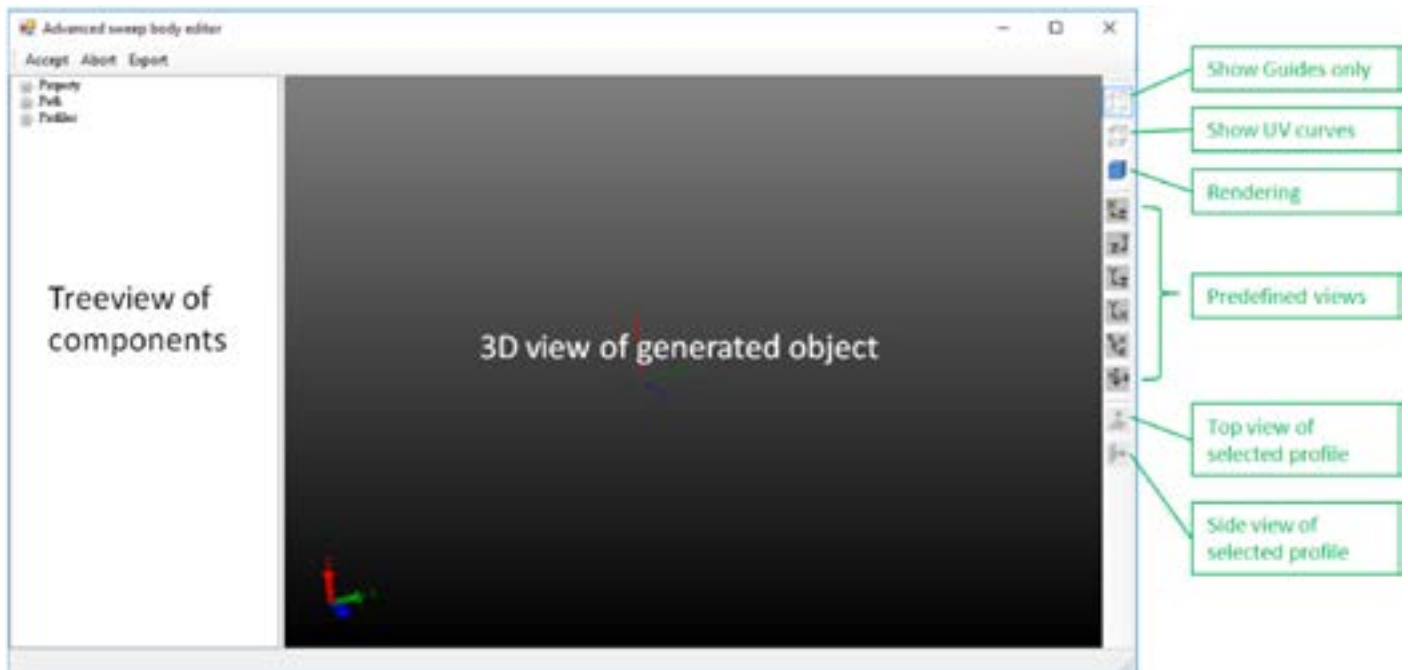
アドバンスドスイープオブジェクトの複雑さのため、オブジェクト追加ウィンドウのタイプオプションでアドバンスドスイープオブジェクトを選択すると、専用のセットアップダイアログが用意されています。

The screenshot shows a software dialog box titled "Add a new object". It contains the following fields and controls:

- Name:** A text input field containing "Object 0".
- Type:** A dropdown menu with "Advanced sweep body" selected.
- Transform:** A section containing:
 - Location:** A text input field containing "(0,0,0)".
 - Local Tilt center:** A text input field containing "(0,0,0)".
 - Tilt then Shift:** A checked checkbox.
 - X angle:** A text input field containing "0" followed by "deg".
 - Y angle:** A text input field containing "0" followed by "deg".
 - Z angle:** A text input field containing "0" followed by "deg".
- Edit Adv. Sweep Body:** A button with a mouse cursor pointing at it.
- Apply** and **Cancel** buttons at the bottom right.

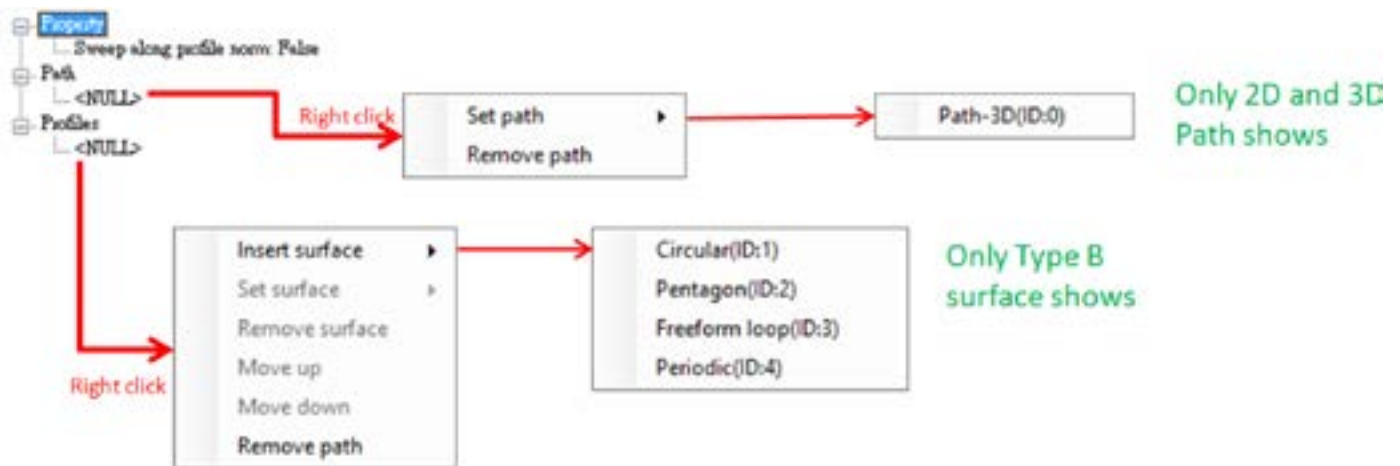
アドバンスドスイープオブジェクトエディタ：

高度なスイープオブジェクトエディタは、高度なスイープオブジェクトを構築するために設計されています。これは、主に 2 つの領域で構成されています。コンポーネントツリービューと、生成されたオブジェクトの 3D ビューです。



右クリックしてツリー表示エリアを選択すると、高度なスイープオブジェクトに必要なコンポーネントを定義するためのポップアップメニューが表示されます。

パスノードまたはプロファイルノードを右クリックすると、異なるメニュー項目が表示されます。



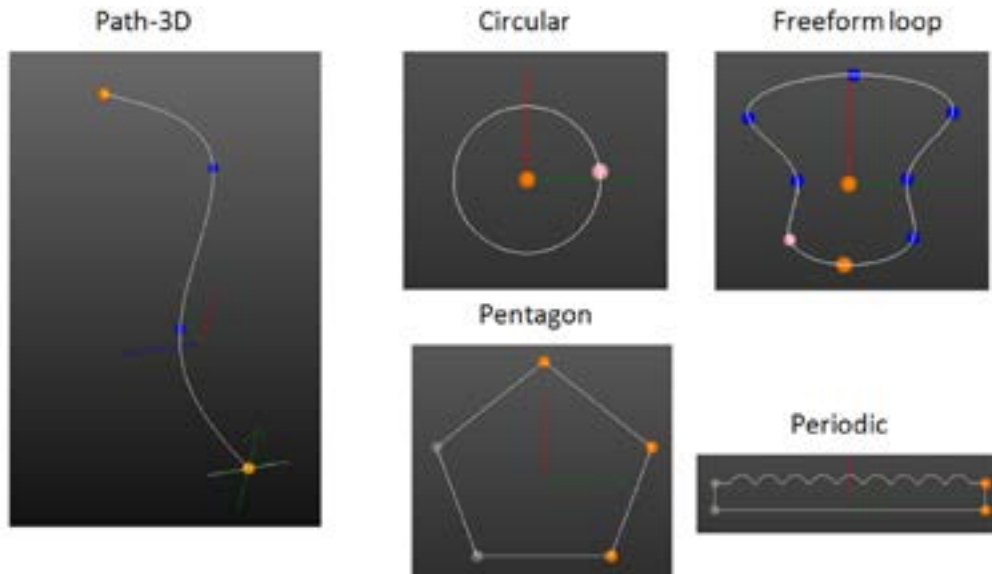
プロファイルの順序は、結果として得られる高度なスイープオブジェクトにとって非常に重要です。順序は、ポップアップメニューの「**Move up**（上に移動）」と「**Move down**（下に移動）」のエントリを使用して編集できます。

各プロファイルは、タイプ B サーフェス、基準平面、スケール X、スケール Y、および相対位置で構成されています。すべての係数が有効または影響可能であるとは限りません。現在のオブジェクトの設定方法によって異なります。

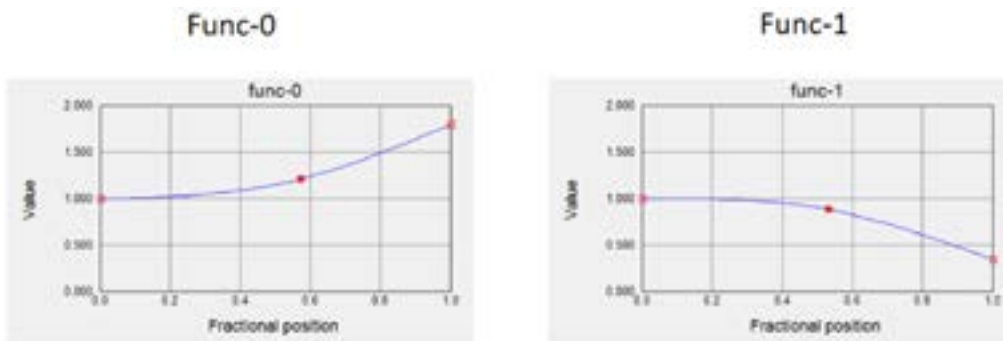
係数	タイプ	定義
Surface	タイプ - B	現在のオブジェクト内でこのプロファイルにリンクされている既存の表面。タイプ B の表面である必要があります。
Origin	ベクトル	プロファイルの位置を示します。原点の位置は、選択されたタイプ B 表面のゼロ点 (0,0) からの対応付け位置を表します。この係数は、「パス」が空の場合のみ編集可能です。パスが設定されている場合、参照平面の原点は、パスルートに沿った入力された分数位置に基づいて計算されます。
Normal	ベクトル	参照平面の法線ベクトルの方向。この係数は、「パス」が空の場合のみ編集可能です。パスが設定されている場合、参照平面の法線は、パスルートに沿った入力された分数位置に基づいて計算されます。
Up	ベクトル	参照平面の Up 方向ベクトルの方向。この係数は、「パス」が空の場合のみ編集可能です。パスが設定されている場合、参照平面の上方向は、パスルートに沿った入力された分数位置に基づいて計算されます。
Scale - X	倍精度浮動小数点数	X 方向の拡大縮小率。X 方向は、タイプ B 表面のローカル Y 方向です。拡大縮小はゼロ点を基準としています。「パス」が設定されており、「パス」プロパティで「パスに沿ってスケール」係数がチェックされている場合、この係数の値に関わらず無視されます。
Scale - Y	倍精度浮動小数点数	Y 方向の拡大縮小率。Y 方向は、タイプ B 表面のローカル Z 方向です。拡大縮小はゼロ点を基準としています。「パス」が設定されており、「パス」プロパティで「パスに沿ってスケール」係数がチェックされている場合、この係数の値に関わらず無視されます。
Fractional position	倍精度浮動小数点数	参照平面をパスの分数位置として割り当てます。これは、「パス」が空でない場合のみ有効になります。

生成されたアドバンスドスイープオブジェクトの例：

例で使用されているサーフェスエレメントのリスト。

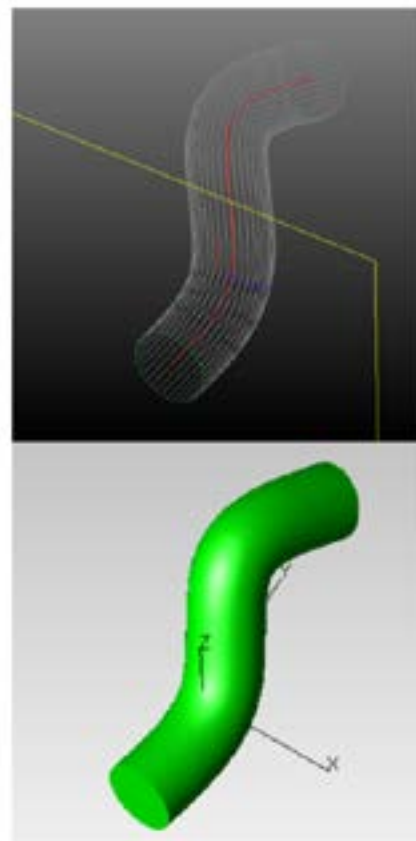


例の中で使用されている関数のリスト



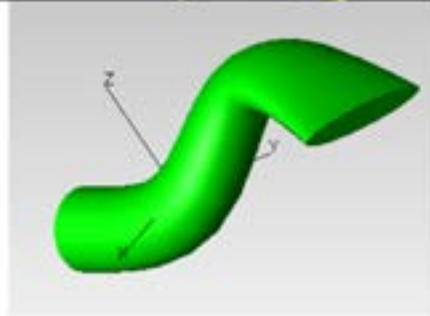
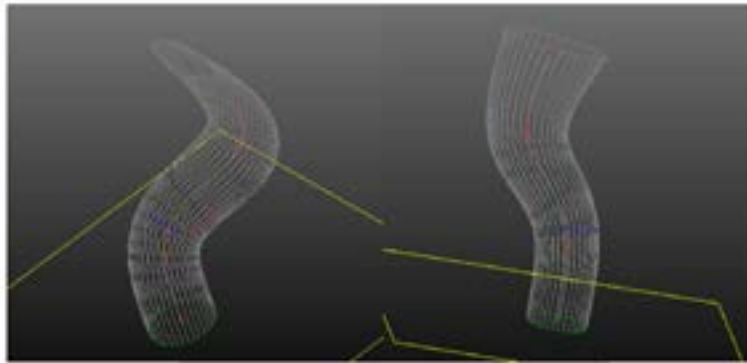
例 1：単純なスイープパスによオブジェクト

- **Path:**
 - Path-3D
- **Profiles:**
 - Circular (1)
- **General Properties:**
 - Normal to profiles: Invalid
- **Path Properties:**
 - Scale along path: **False**
 - Path scale X: Invalid
 - Path scale Y: Invalid
- **Profile Properties:**
 - (1)Scale X: 1.0
 - (1)Scale Y: 1.0



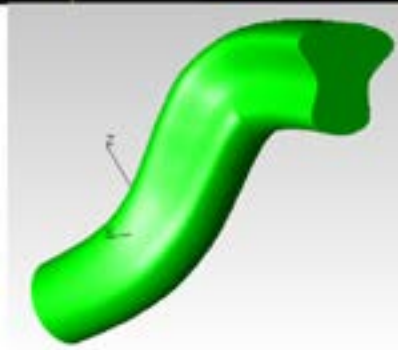
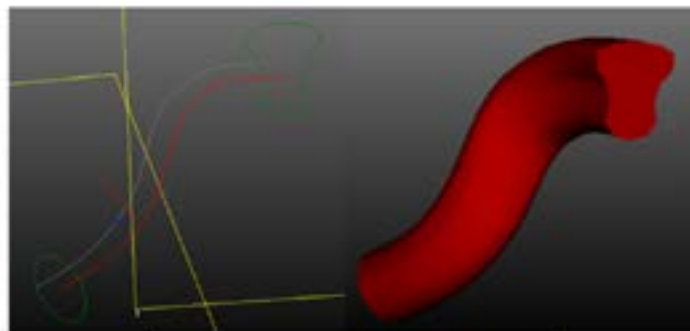
例 2：不等スケーリングによるスイープ

- **Path:**
 - Path-3D
- **Profiles:**
 - Circular (1)
- **General Properties:**
 - Normal-to-profiles: invalid
- **Path Properties:**
 - Scale along path: True
 - Path scale X: func-0
 - Path scale Y: func-1
- **Profile Properties:**
 - (1)Scale X: invalid
 - (1)Scale Y: invalid



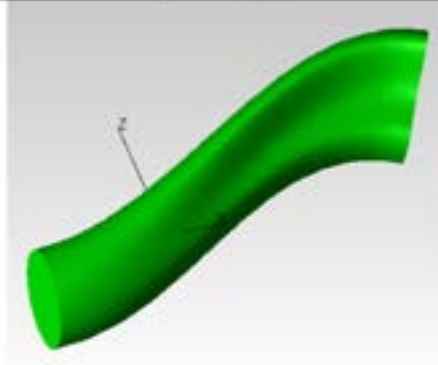
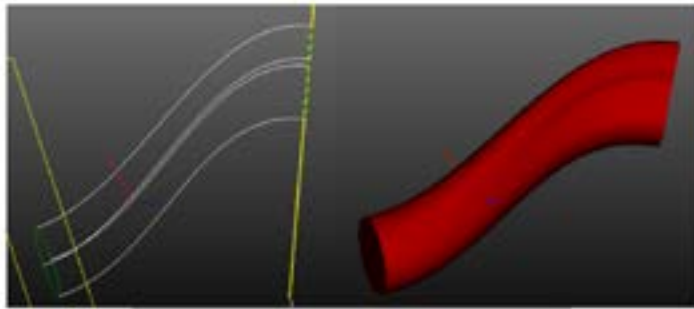
例 3：異なる開口形状を持つスイープパイプ

- **Path:**
 - Path-3D
- **Profiles:**
 - Circular (1)
 - Freeform loop (2)
- **General Properties:**
 - Normal-to-profiles: invalid
- **Path Properties:**
 - Scale along path: False
 - Path scale X: invalid
 - Path scale Y: invalid
- **Profile Properties:**
 - (1)Scale X: 1.0
 - (1)Scale Y: 1.0
 - (2)Scale X: 1.0
 - (2)Scale Y: 1.0



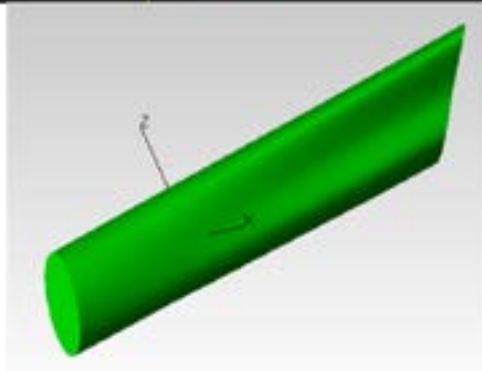
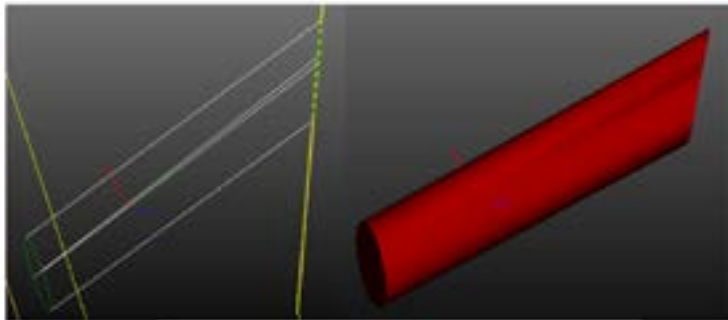
例 4 : "Normal to profiles" を ON にしたスイープ

- **Path:**
 - <NULL>
- **Profiles:**
 - Circular (1)
 - Freeform loop (2)
- **General Properties:**
 - Normal to profiles: True
- **Path Properties:**
 - Scale along path: False
 - Path scale X: Invalid
 - Path scale Y: Invalid
- **Profile Properties:**
 - (1) Scale X: 1.0
 - (1) Scale Y: 1.0
 - (2) Scale X: 1.0
 - (2) Scale Y: 1.0



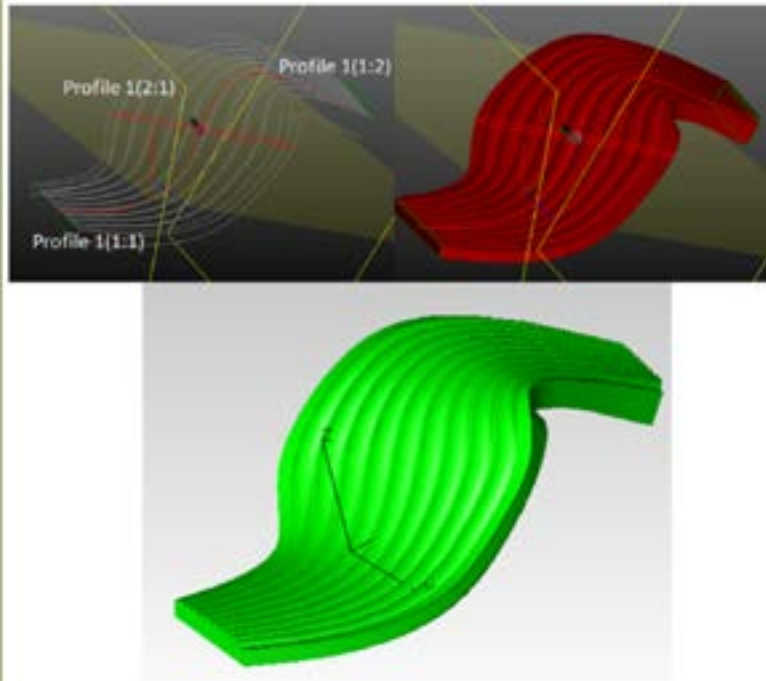
例 5 : "Normal to profile" を Off にしたスイープ

- **Path:**
 - <NULL>
- **Profiles:**
 - Circular (1)
 - Freeform loop (2)
- **General Properties:**
 - Normal to profiles: False
- **Path Properties:**
 - Scale along path: False
 - Path scale X: Invalid
 - Path scale Y: Invalid
- **Profile Properties:**
 - (1) Scale X: 1.0
 - (1) Scale Y: 1.0
 - (2) Scale X: 1.0
 - (2) Scale Y: 1.0



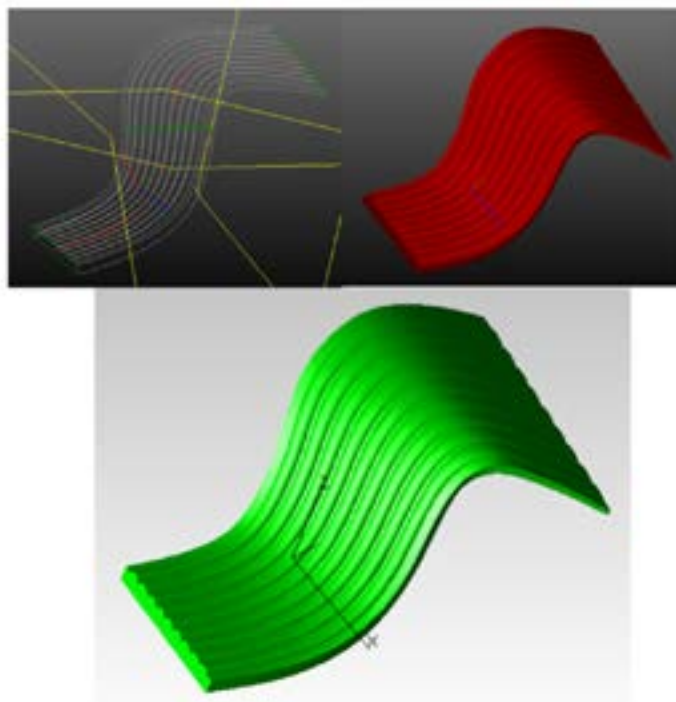
例6：異なるスケールのプロファイルでスイープ

- **Path:**
 - Path-3D
- **Profiles:**
 - Periodic (1)
 - Periodic (2)
 - Periodic (3)
- **General Properties:**
 - Normal-to-profiles: ~~False~~
- **Path Properties:**
 - Scale along path: ~~False~~
 - Path scale X: ~~Invalid~~
 - Path scale Y: ~~Invalid~~
- **Profile Properties:**
 - (1)Scale X: 1.0
 - (1)Scale Y: 1.0
 - (2)Scale X: 2.0
 - (2)Scale Y: 1.0
 - (3)Scale X: 1.0
 - (3)Scale Y: 2.0



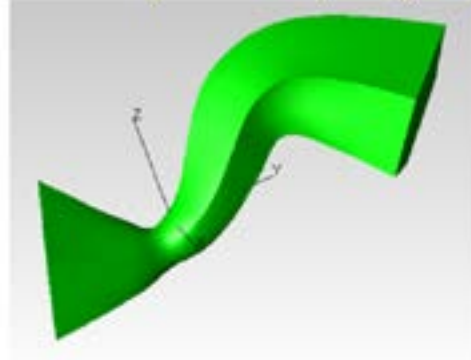
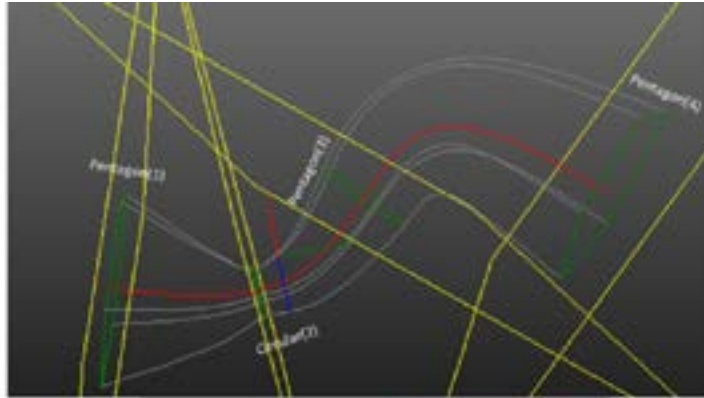
例7：関数スケールによるスイープ

- **Path:**
 - Path-3D
- **Profiles:**
 - Periodic (1)
 - Periodic (2)
 - Periodic (3)
- **General Properties:**
 - Normal-to-profiles: ~~False~~
- **Path Properties:**
 - Scale along path: ~~True~~
 - Path scale X: func-0
 - Path scale Y: func-1
- **Profile Properties:**
 - (1)Scale X: ~~1.0~~
 - (1)Scale Y: ~~1.0~~
 - (2)Scale X: ~~2.0~~
 - (2)Scale Y: ~~1.0~~
 - (3)Scale X: ~~1.0~~
 - (3)Scale Y: ~~2.0~~



例 8：円形から多角形へのスweep

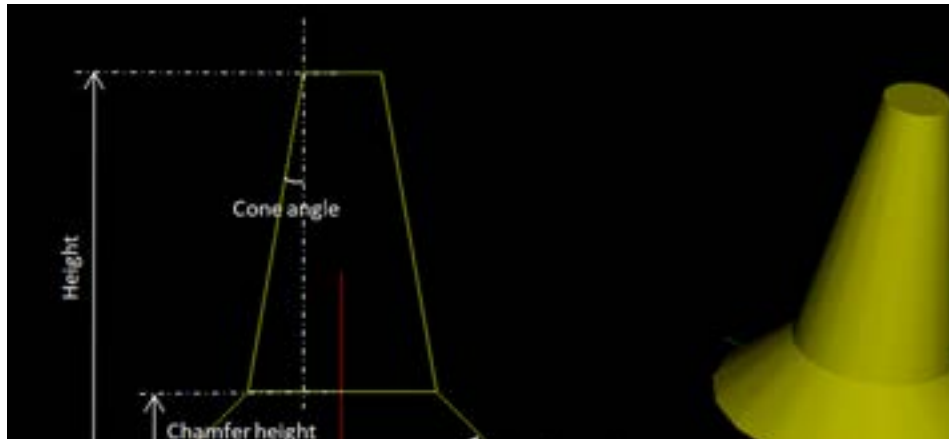
- **Path:**
 - Path-3D
- **Profiles:**
 - Pentagon (1)
 - Circular (2)
 - Pentagon (3)
 - Pentagon (4)
- **General Properties:**
 - Normal to profiles: **False**
- **Path Properties:**
 - Scale along path: **False**
 - Path scale X: 1.0
 - Path scale Y: 1.0
- **Profile Properties:**
 - (1)Scale X: 1.0
 - (1)Scale Y: 1.0
 - (2)Scale X: 0.5
 - (2)Scale Y: 0.5
 - (3)Scale X: 0.5
 - (3)Scale Y: 0.5
 - (4)Scale X: 1.0
 - (4)Scale Y: 1.0



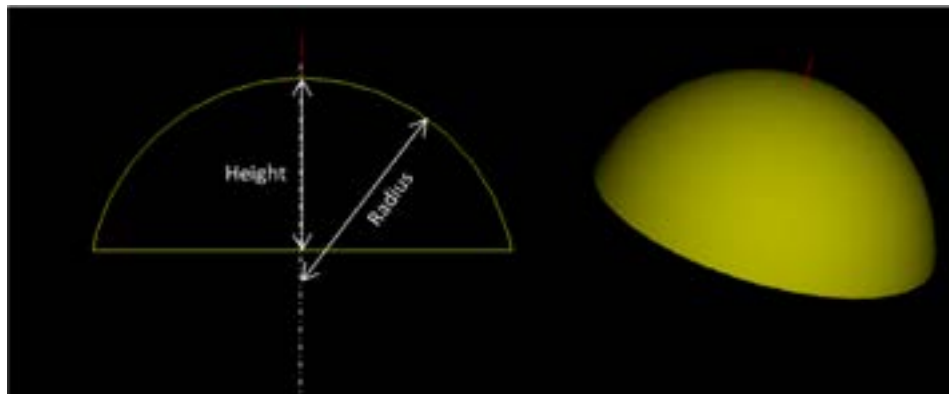
オブジェクトタイプ：レプタイルオブジェクト

レプタイルオブジェクトは、TracePro Expert エディションでサポートされている定義済みのレプタイル形状を実際のオブジェクトとして扱います。サポートされている幾何学的形状には、円錐、球、寄棟屋根、楕円体、ログ（丸太）、拡張プリズム、扁平円錐、尖頭円錐、トーラス、ポリゴン、ピラミッド、DMD、ブロック、および彫刻ログが含まれます。すべてのレプタイル形状は定義済みであるため、レプタイルオブジェクトはサーフェスを必要とせず、オブジェクトエディタで係数を入力するだけで形状を制御できます。唯一の例外は非球面であり、非球面はオブジェクトではなく面であるためです。

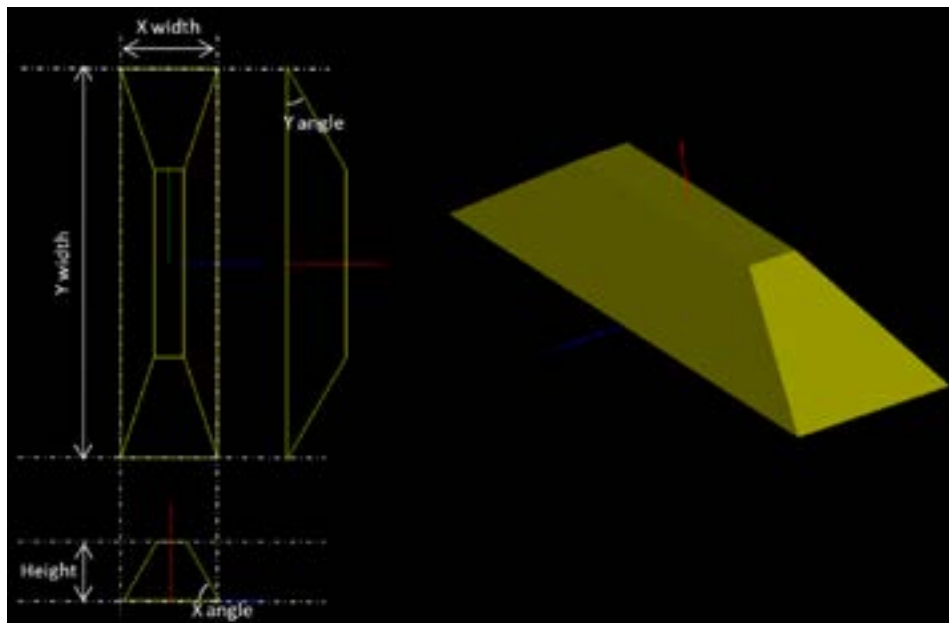
コーン：



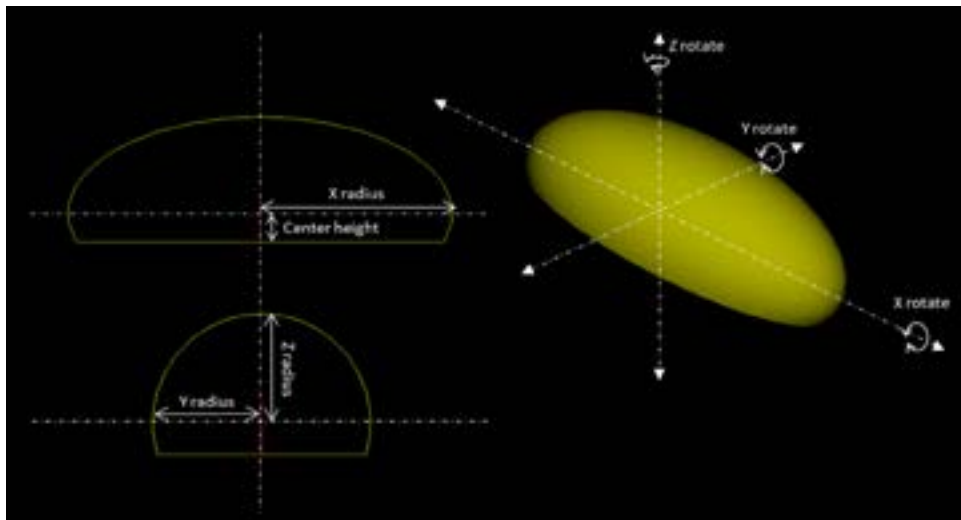
球：



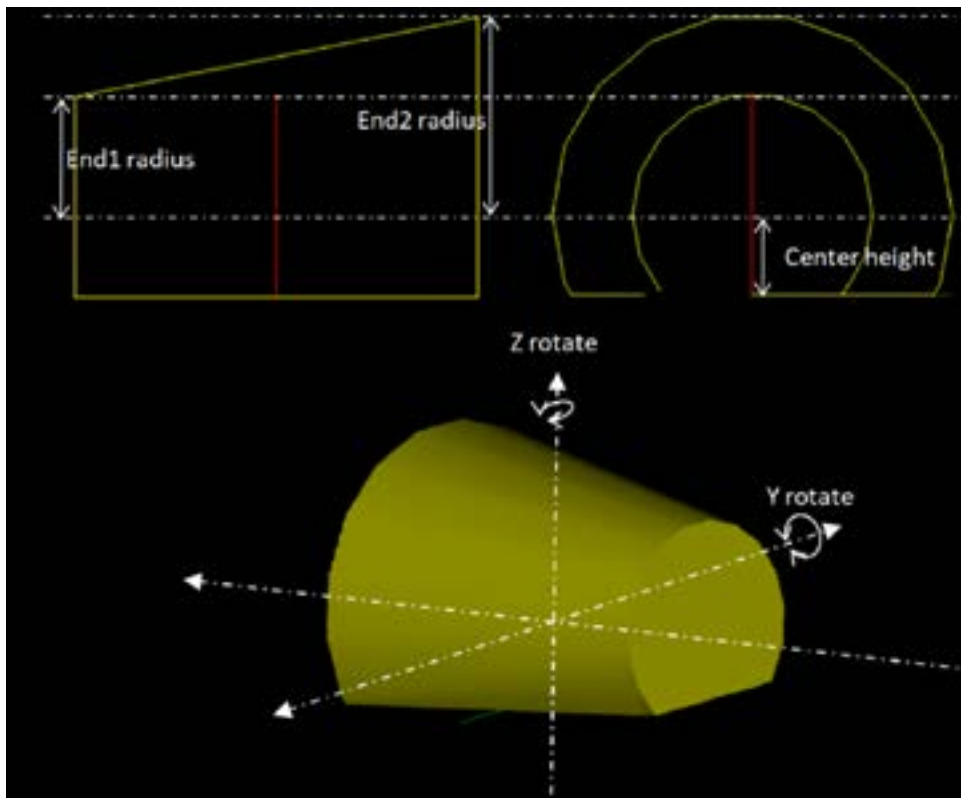
ヒップルーフ：



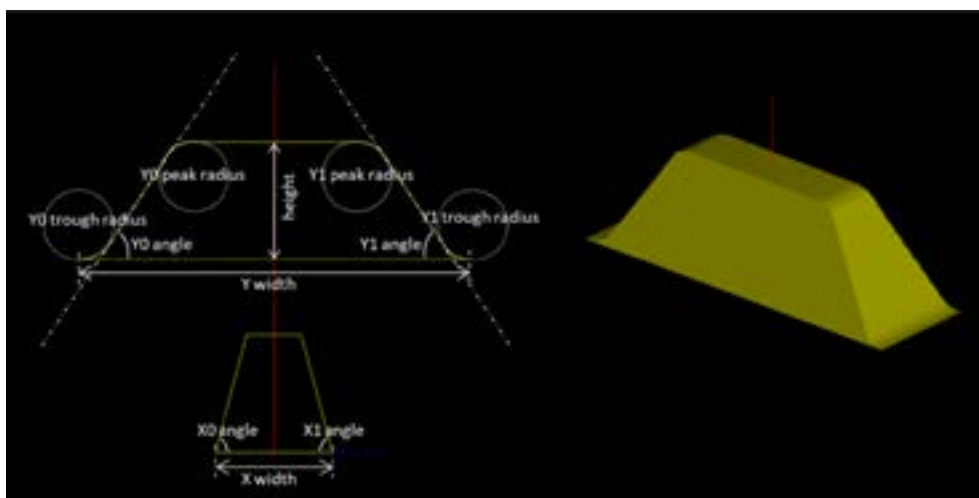
エリプソイド：



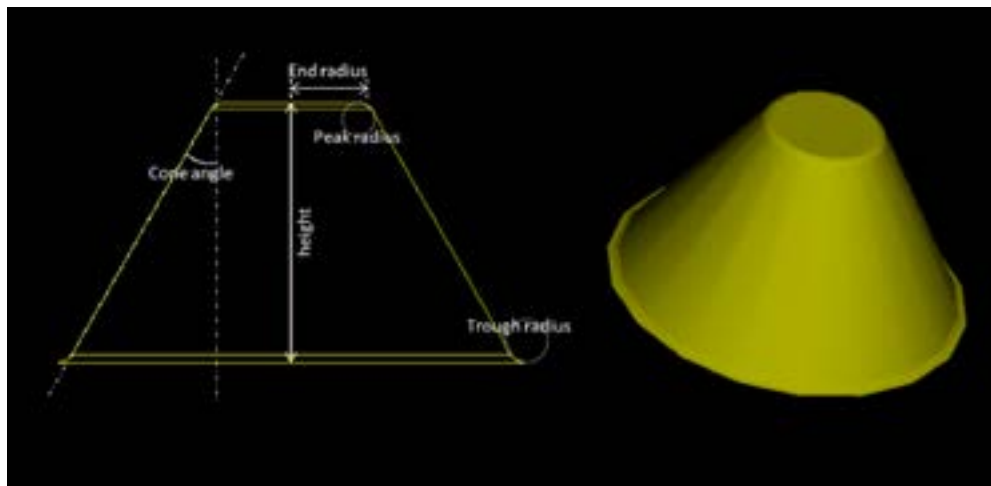
ログ：



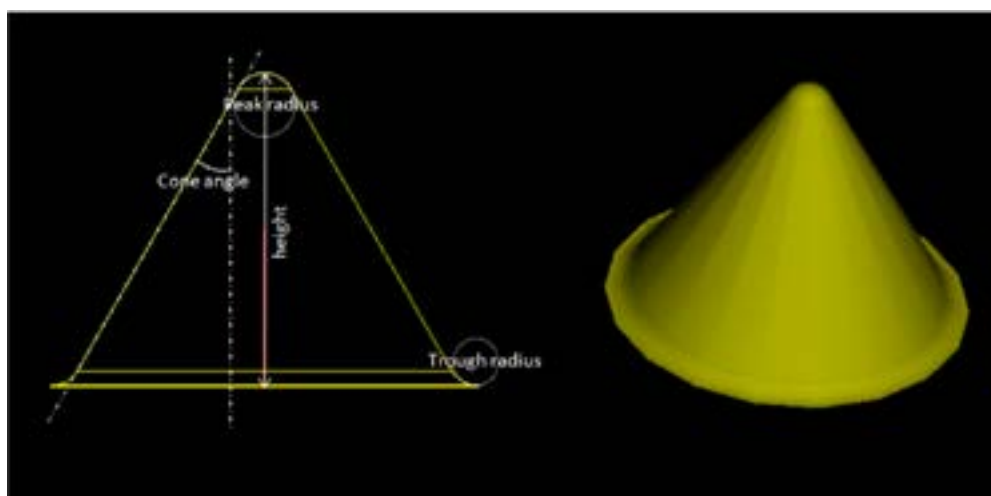
エンハンスプリズム：



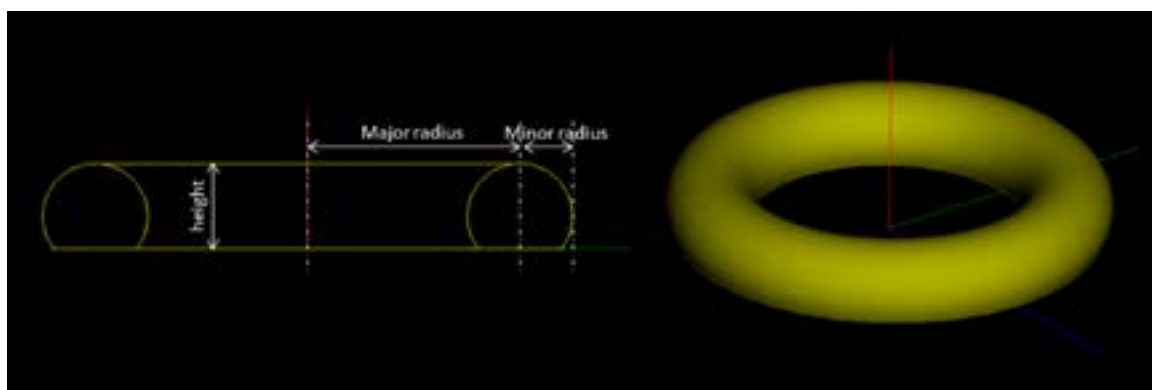
頂点を平たん化したコーン：



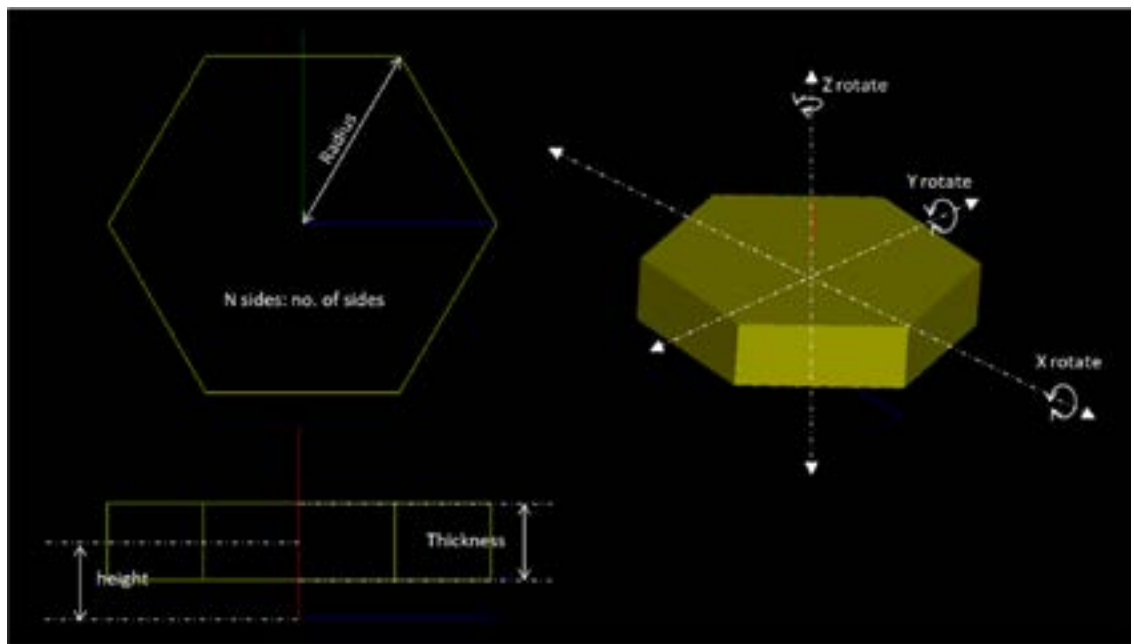
先端が r を持つコーン：



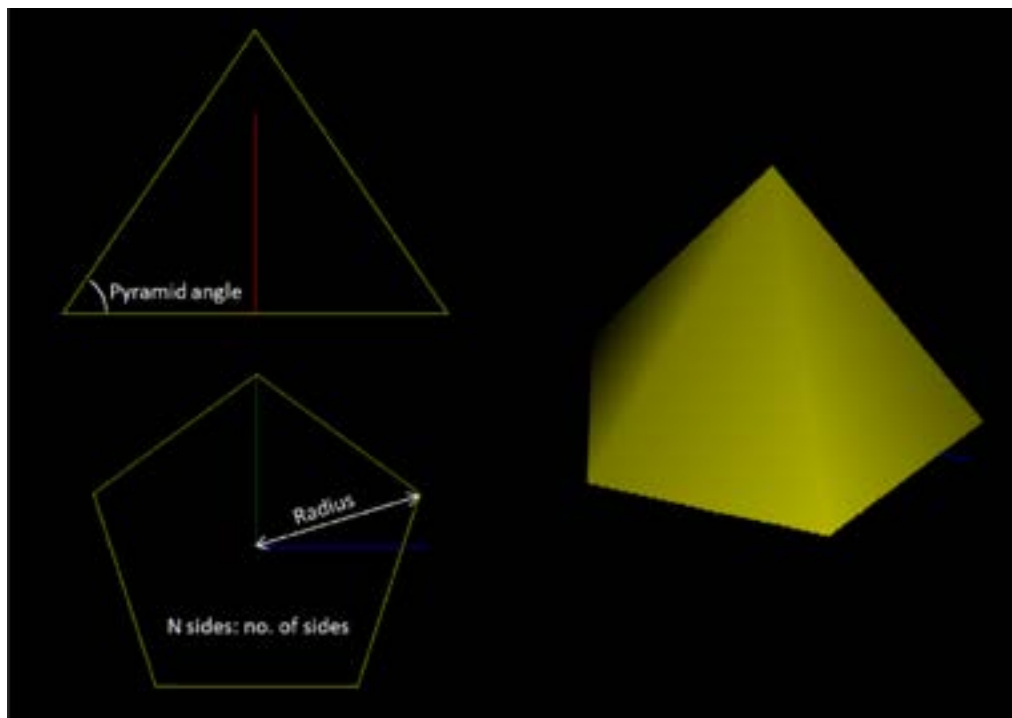
トーラス：



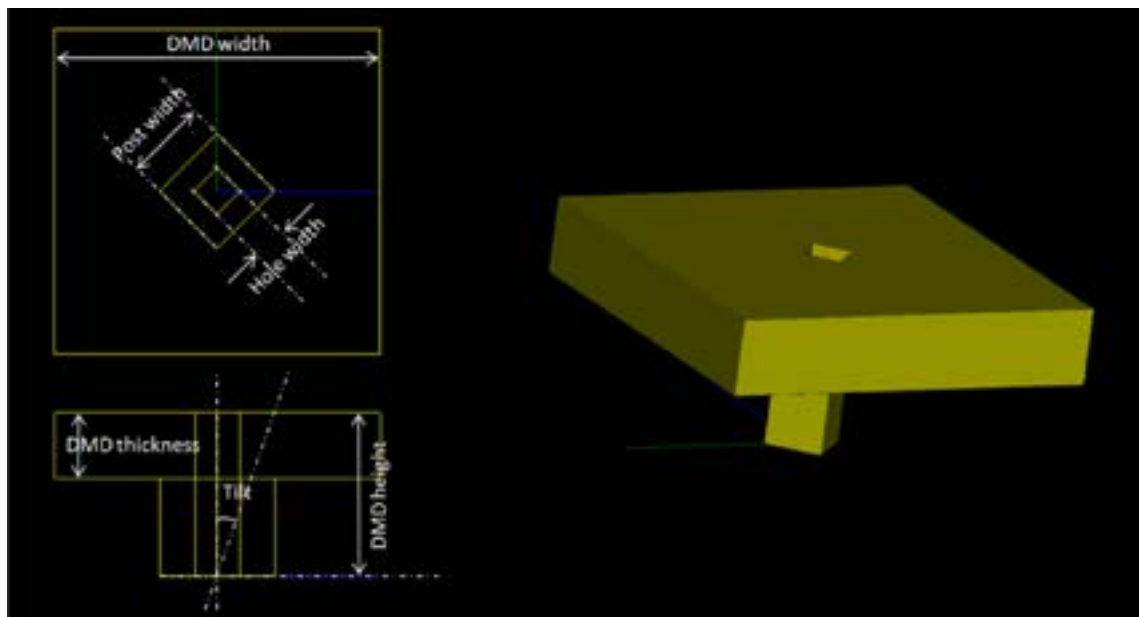
ポリゴン：



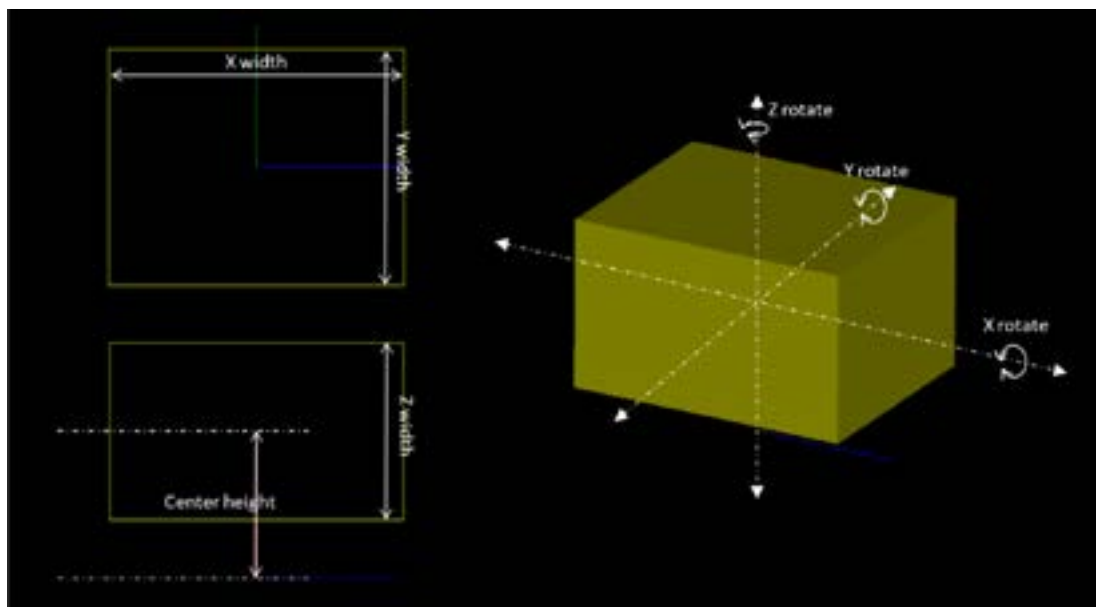
ピラミッド：



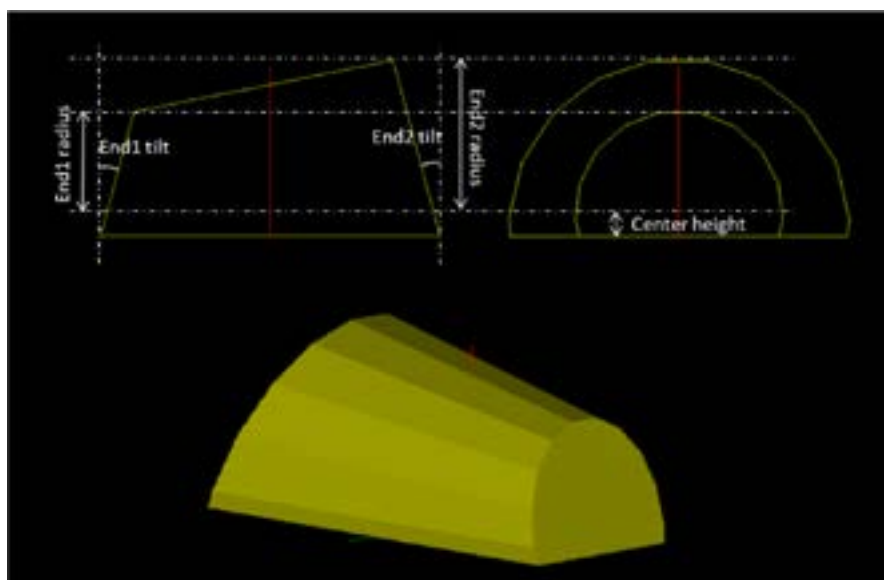
DMD :



ブロック :



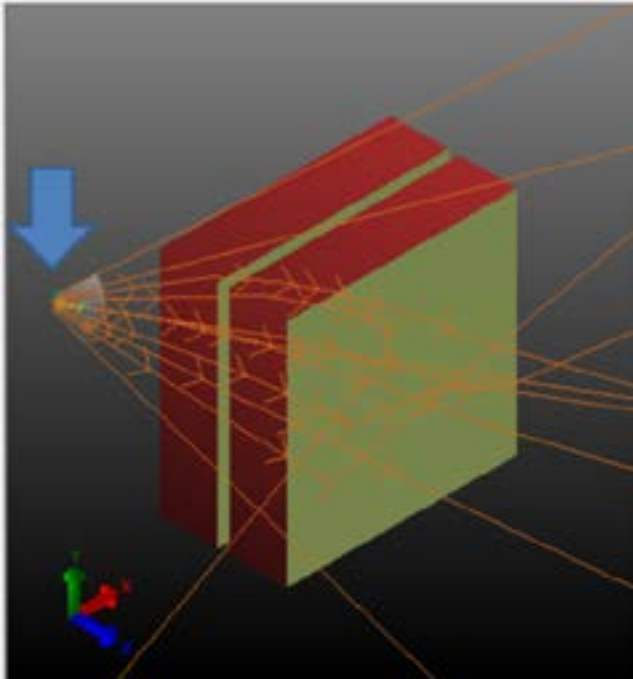
チゼルドログ :



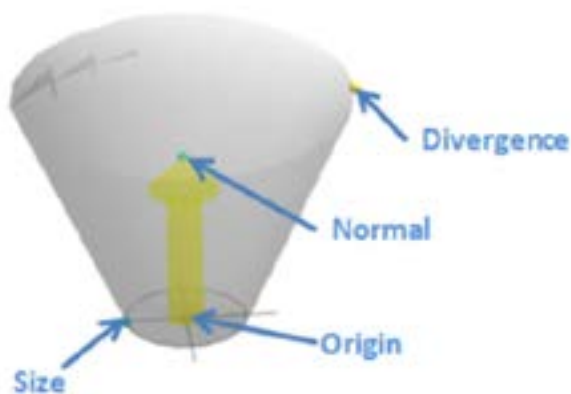
ソースの定義

ソースを（インタラクティブオプティマイザにおいて）定義するということは、設計を評価するためのリアルタイムな光線追跡を実行する際に使用する単一のソースを設定することを意味します。これは、設計の性能を視覚化するための強力なツールです。

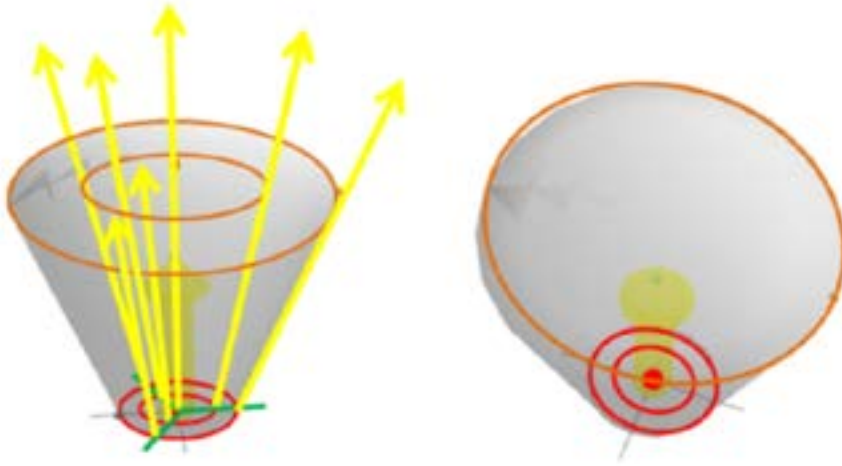
インタラクティブオプティマイザでは、定義された単一のソースを用いて、設計の変更が光の挙動にどのように影響するかをリアルタイムに確認できます。これにより、設計者は効率的に照明システムや光学系の性能を評価し、最適化することができます。



インタラクティブオプティマイザのソースは基本的に光のコーン（円錐）です。その位置、方向、サイズ、発散角を設定することで、ソースは定義されたコーン内に一定数の光線を放出します。下図は各パラメータに対応する制御点を示しており、それらはすべてオブジェクトビューア内でインタラクティブに操作できます。ソースの係数を変更するもう一つの方法は、プロパティウィンドウを使用することです。オブジェクトビューアでソースが選択されると、そのすべてのパラメータがプロパティウィンドウに更新され、変更できるようになります。



放射される光線数は、スライス数とリング数の積によって決定されます。以下の図は、スライスとリングがそれぞれ3の場合に、ソースがどのように分割されるかを示しています。



1. Origin (原点) - ソースの中心の x、y、z 座標。
2. Normal (法線) - 光線方向を定義するための x、y、z 方向余弦。
3. Source size (ソースサイズ) - 光線が生成されるソースの範囲を指定します。
4. Divergence angle (発散角) - 光線が生成される円錐の半角。
5. Rings & Slices (リングとスライス) - リングとスライスによって形成される各セグメントから 1 本の光線が開始されます。

ソースの作成はオブジェクトビューアから開始されます。まず、ツールバーのアイコン () を押して、オブジェクトビューアをソースモードに切り替えます。ソースモードでは、3D ビューのポップアップメニューに追加の項目「Define source (ソースを定義)」が表示されます。

ポップアップメニューを開いたマウスの位置がソースのデフォルトの位置ですが、「Property Editor (プロパティエディタ)」ウィンドウで変更できます。

ソースは、原点、法線、サイズ、発散、リングとスライスの 6 つの係数で構成されています。模式図と説明については、「Light source (ソース)」のセクションを参照してください。ソースのパラメータを編集するには、ソースの任意のコントロールポイントをクリックして「Property Editor (プロパティエディタ)」を開きます。

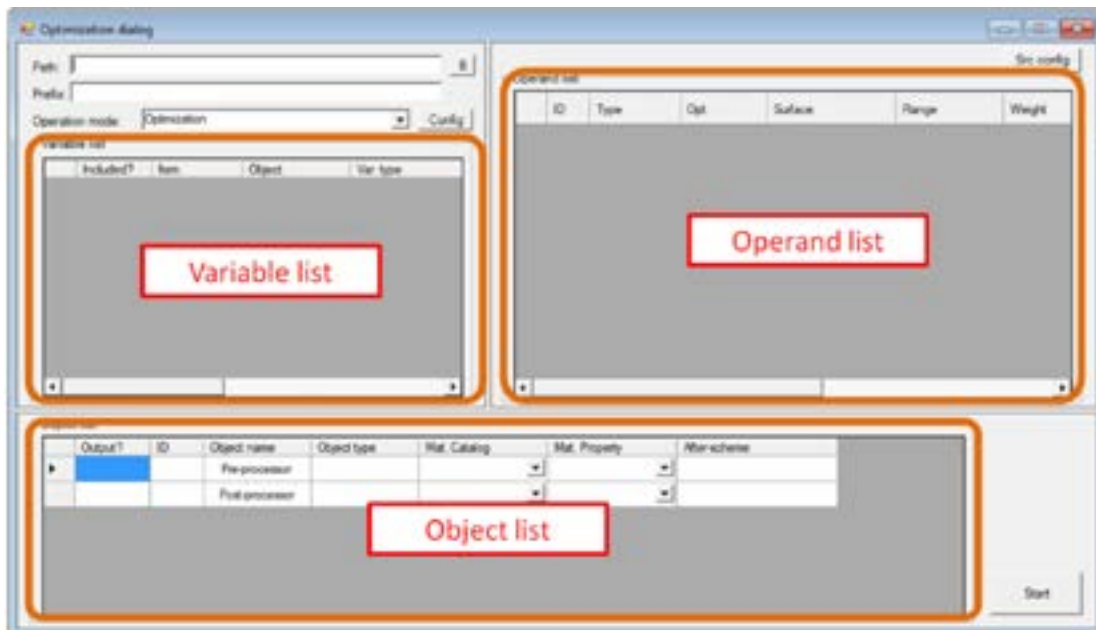
Property editor				
Description	Value	Type	Lower limit / Pickup	Upper limit
Origin	(0.8233338,-1.209419,-2.032738)			
Normal	(-0.3114442,0.4979248,0.809366)			
Source size	0.6			
Divergence (deg)	10			
Rings	1			
Slices	5			

オプティマイゼーション

最適化ウィンドウ

適切な変数が設定されると、インタラクティブオプティマイザは最適化に進むことができます。変数を変化させると、結果として得られる形状が変わり、TracePro で評価される光学性能も変化します。性能が向上した場合、オプティマイザは同じ方向に変数を変化させ続け、性能が低下した場合、オプティマイザは反対方向に変数を変化させます。

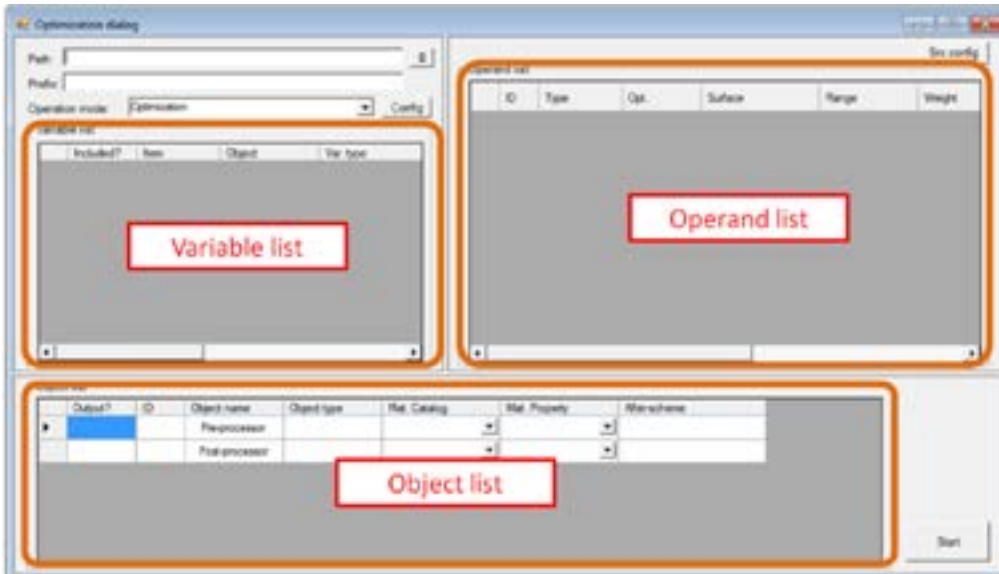
最適化ウィンドウを開くには、インタラクティブオプティマイザのメインメニューから「最適化」項目を選択します。最適化ウィンドウには、変数リスト、オペランドリスト、オブジェクトリストの3つのコンポーネントがあります。



詳細

- Optimization settings: 最適化設定
- Configuration panel: 構成パネル、設定パネル
- Variable list: 変数リスト
- Variables: 変数
- Src config: ソース構成、ソース設定
- Operand: オペランド
- Object list: オブジェクトリスト
- Optimization algorithm: 最適化アルゴリズム

Optimization settings: 最適化の設定



Path (パス)

最適化処理のログファイルが保存される指定される場所です。最適化ログウィンドウでこれらのファイルを表示できます。

Prefix: (プレフィックス)

プレフィックスを使用すると、同じフォルダ内の異なる最適化からのログファイルを区別するのに役立ちます。

Config button (設定ボタン)

ダウンスヒル・シンプレックス法の場合、インタラクティブオブティマイザはアルゴリズムの制御に関するいくつかのオプションを提供します。設定ボタンをクリックすると、設定ウィンドウが開きます。設定パネルの詳細については、「[Configuration for Downhill simplex algorithm \(ダウンスヒル・シンプレックス法の構成\)](#)」を参照してください。

Optimization modes: (最適化モード)

コンボボックスで選択可能なオプションは、「[Optimization algorithms \(最適化アルゴリズム\)](#)」のセクションに記載されています。

Variable list (変数リスト)

定義されたすべての変数をリスト表示します。ユーザー定義変数をここに追加することもできます。

Operand list (オペランドリスト)

最適化中の性能評価に使用するオペランド項目を挿入、移動、削除します。

Object list (オブジェクトリスト)

すべてのオブジェクトを表示し、オブジェクトのプロパティ定義、TracePro でのオブジェクト作成、アフタースキームコマンドの編集 / テスト、または最適化の 1 回のイテレーションの実行を可能にします。

Src config (ソース設定)

複数の構成を定義します。「[Source Configuration. \(ソース設定\)](#)」を参照してください。

Start button (開始ボタン)

最適化を開始します。

Downhill Simplex algorithm（ダウンヒルシンプレックスアルゴリズム）の設定パネル

Configuration for Downhill Simplex algorithm

Random seed = 1000

Characteristic length

☒ Determined by ratio of limits Ratio= 0.5

☐ Determined by length Value= 0.1

Stop condition

Maximum iterations number: 5000

Iteration tolerance: 0.0001

Model refresh after optimization

☒ Start point ☐ End point ☐ Best point

OK Cancel

Random seed ランダムシード	初期シンプレックスを構築するための乱数列を生成する際のシードとして使用される整数。一意のシード番号は一意の乱数列を決定するため、シード番号を制御することで再現可能な最適化プロセスを作成できます。
Characteristic length 特性長	初期シンプレックスは下り坂シンプレックス法の開始点であるため、最終結果がどこに向かうか、および最適化中にパスがどのように作成されるかを決定します。初期シンプレックスは、N 変数の場合、N+1 個の頂点を持つシンプレックスです。すべての頂点はランダムプロセスによって生成された変数のセットであり、特性長はその乱数の範囲です。特性長には、「制限の比率によって決定」と「長さによって決定」の 2 つのオプションがあります。前者は、各変数の入力制限の指定された比率によって乱数の境界を計算する方法です。後者は、指定された値を乱数の境界として使用する方法です。
Stop condition 停止条件	最適化プロセスをいつ停止するかを決定するオプションです。2 つのパラメータが両方利用可能であり、最初に満たされた方に応じて停止条件がトリガーされます。最適化中の許容誤差の計算は、次のルールに従います。 - 最高誤差値と最低誤差値の絶対値の合計が 1e-10 未満の場合、許容誤差はゼロに設定されます。最適化プロセスは停止します。 - 上記条件が満たされない場合、許容誤差は $2 * \text{abs}(\text{err}(\text{hi}) - \text{err}(\text{lo})) / (\text{abs}(\text{err}(\text{hi})) + \text{abs}(\text{err}(\text{lo})))$ によって計算されます。
Model refresh after optimization 最適化後のモデル更新	最適化が停止したとき（終了理由に関わらず）、モデルビューを更新するための 3 つのオプションがあります。「開始点」、「終了点」、「最良点」です。

解説

このテキストは、最適化アルゴリズム（特に下り坂シンプレックス法に関連する）の設定項目について説

明しています。それぞれの項目が最適化プロセスにどのように影響するかを理解するのに役立ちます。

ランダムシード: 最適化の初期段階で使用する乱数を制御します。同じシード値を使用すると、毎回同じ初期状態から最適化が始まるため、結果を再現することができます。これは、アルゴリズムの動作を理解したり、異なる設定の効果を比較したりする際に重要です。

特性長: 下り坂シンプレックス法の初期シンプレックス（探索の開始点となる多面体）の大きさを決定します。

制限の比率によって決定: 各変数の定義された範囲（上限と下限）に対して、指定された比率で乱数の範囲を決定します。例えば、範囲の 10% を特性長とする場合、各変数の初期値は、その変数の範囲内の 10% の幅でランダムに生成されます。

長さによって決定: 特定の値を乱数の範囲として直接使用します。この場合、各変数の初期値は、その変数の現在値を中心に、指定された長さの範囲内でランダムに生成されます。初期シンプレックスの形状と位置は、最適化の効率と最終的な解に影響を与える可能性があります。

停止条件: 最適化アルゴリズムがいつ探索を終了するかを決定します。ここでは、誤差の範囲に基づいて停止する条件が説明されています。

- 非常に小さな誤差範囲（ $1e-10$ 未満）になった場合、これ以上改善が見込めないと判断し、許容誤差をゼロにして停止します。
- それ以外の場合は、最高誤差と最低誤差の差と平均に基づいて許容誤差を計算し、この許容誤差が十分に小さくなった場合に停止すると考えられます。

最適化後のモデル更新: 最適化が終了した後に、どの時点のモデルの状態を画面に表示するかを選択します。

開始点: 最適化を開始した時点のモデルを表示します。

終了点: 最適化が終了した時点のモデルを表示します。これは、アルゴリズムが見つけた最終的な解に対応します。

最良点: 最適化の全過程で最も良い評価値（誤差が最も小さい）を示した時点のモデルを表示します。終了点が必ずしも最良点とは限りません。

これらの設定を適切に調整することで、特定の最適化問題に対して、より効率的で信頼性の高い結果を得ることが期待できます。

Variable list (変数リスト)

すべての事前定義変数は変数リストに表示されます。変数リストを右クリックすると、ユーザー定義変数の挿入および削除コマンドにアクセスするためのポップアップメニューが開きます。



Included?	Item	Object	Var. type	Value	Lo. limt.	Hi. limt.
☒	Thickness	Object 0/Object...	RelativeVariable	5	0	0
☒	Position-Z	Ctrl Fnt.2@Seg...	RelativeVariable	4.160217285156...	0	0

User-defined variable (ユーザー定義変数) :

あらかじめ定義されたサーフェスに関連しない変数、例えばスキームコマンドや after-scheme (アフタースキーム) プログラムで使用される他の値によってオブジェクトが新しい位置に移動される場合、ユーザー定義変数はこれらの値を最適化する方法を提供します。

通常の変数と同様に、ユーザー定義変数も絶対モードと相対モードの間で切り替えることができます。これら二つのモードの意味は、「変数」で説明されている通常の変数の場合と全く同じです。

アフタースキームプログラミングでユーザー定義変数を使用するには、var() 関数を使用します。詳細な説明は「[After-scheme programming \(アフタースキームプログラミング\)](#)」にあります。

Variables（変数）

デザインを最適化するためには、設計目標を達成するためにどのパラメータを変化させることができるかを指定する必要があります。最も一般的な変数は、制御点の座標です。制御点を移動させることで、結果として得られるプロファイルが変化し、それが今度はオブジェクトを変化させます。パラメータ化された数式から生成されるオブジェクト、すなわちバイコーニック（タイプ A） およびコーン（タイプ B）の場合、変化させる制御点はありませんが、コーニック常数（半径と円錐定数）を変数として指定できます。

変数は、次の 3 つのタイプに分類されます。

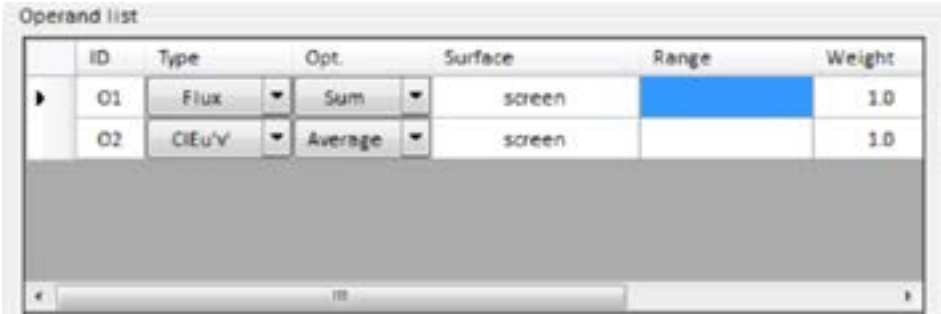
Specified (指定):	パラメータは定数であり、最適化中に変動しません。
Relative variable (相対変数):	パラメータの変動範囲は、その現在の値を基準とします。例えば、開始値が 0.3、上限が 0.3、下限が 0.2 の場合、変数の許容範囲は 0.1 から 0.6 になります。
Absolute variable (絶対変数):	上限と下限は、現在の値の高い側と低い側です。開始値は、現在の値が変数の範囲内にある場合は現在の値、現在の値が範囲外にある場合は範囲の midpoint になります。

Operand（オペランド）

インタラクティブオブティマイザは、最適化中のシステム性能を評価するためにオペランドを使用します。オブティマイザは、次の最適化を決定する際にオペランドの値を最小化することを目標とするため、オペランドのより小さい値がより良い性能を表すように設定することが非常に重要です。

新しいオペランドを追加するには、オペランドリスト内で右クリックし、「挿入」を選択します。「削除」、「上へ移動」、「下へ移動」オプションもポップアップメニューにあります。

オペランドは光学システムの性能を数値で表すための重要な要素であり、インタラクティブオブティマイザはこの数値を最小化することで、設計者が求める最適なシステム構成を見つけ出す手助けをします。



ID	インタラクティブオブティマイザは、最適化中にシステムの性能を評価するためにオペランドを使用します。オブティマイザは、次の最適化を決定する際にオペランドの値を最小化することを目標とするため、オペランドの小さい値がより良い性能を表すことが非常に重要です。新しいオペランドを追加するには、オペランドリスト内で右クリックし、「挿入」を選択します。「削除」、「上へ移動」、「下へ移動」オプションもポップアップメニューにあります。
Type タイプ	すべてのオペランドには一意の識別番号があります。IDの主な目的は、ユーザー定義オペランドを定義する際にオペランドを参照することです。
Option オプション	選択したオペランドのタイプに複数のオプションがある場合、このセルに表示されます。
Surface サーフェス	ほとんどのオペランドでは、照度マップなどの解析データを評価するターゲットとしてサーフェスを指定する必要があります。正しいサーフェスが評価されるように、サーフェス名は TracePro モデル内で一意である必要があります。あるいは、「オブジェクト名 / サーフェス名」のようにスラッシュで区切ってオブジェクト名とサーフェス名を一緒に指定することもできます。いずれの場合も、ターゲットを指定する文字列はサーフェスを一意に識別する必要があります。
Range 範囲	一部のオペランドは範囲セルを使用し、一部は使用しません。追加の範囲セクターが必要なオペランドは、フラックス、CI Eu^v、CI Exy です。範囲セクターは、現在のオペランドの範囲を指定するためのグラフィカルユーザーインターフェイスを備えたダイアログです。範囲のタイプは「マップ全体」、「領域」、「ポイント」であり、選択したオペランドのプロパティに応じて変化します。「領域」モードでは、オペランドが計算される領域を定義するために長方形をドラッグします。「ポイント」モードでは、照度が評価される位置を指定するためにマップ上の点をクリックします。中央の黒いマップは、正規化されたスケール（つまり、マップの座標は x 軸と y 軸に沿って -0.5 から 0.5 まで）で TracePro から得られた照度を表します。

Weight 重み	オペランドリストの最終値は、すべての個々のオペランド誤差の二乗平均平方根を使用して計算されます。各オペランドに異なる重みを使用して、寄与のバランスを取り、設計目標の達成を支援できます。
T a r g e t value 目標値	目標値は、現在のオペランドの望ましい値です。最適化は、すべてのオペランドの重み付き二乗平均平方根誤差を最小化します。各オペランドの誤差は、誤差 = 実測値 - 目標値 として計算されます。各オペランドタイプには異なる計算があります。各タイプの説明は、対応するセクション（オペランド）にあります。」マップオプション TracePro では、カンデラマップと照度マップには、結果のプロットを制御するためのオプションがあります。マップオプションは、誤差値の次の計算のためにここで事前に定義できます。」
S o u r c e config ソース構成	複数の光源を含むシステムの場合、システム全体の評価には、一部の光源をオンにしたり、いくつかの光学部品を無効にしたりするなど、異なる構成が必要です。ソース構成は光源の定義だけでなく、オペランドが評価される前に実行できる一連のスキームのように機能します。

Object list(オブジェクトリスト)

オブジェクトのプロパティを定義するには、オブジェクトリストを使用します。すべての事前定義済みオブジェクトがここに表示されますが、最初の行を除き、各行の最初の列にあるチェックボックスをオンまたはオフにすることで、それらを TracePro に送信するかどうかを決定できます。最初の行は、オブジェクトを TracePro に送信する前に実行される after-scheme (アフタースキーム) コマンドを割り当てるために使用されます。

リスト内を右クリックすると、最適化モデルが正しく定義されていることを確認するのに役立つコマンドのメニューが表示されます。「オブジェクトを作成」を使用して、オブジェクトが TracePro で正しく作成されていることを確認します。アフタースキームコマンドをテストします。「1 回反復実行」を使用して最適化モデルを検証します。1 回反復を実行すると、結合されたエラー値ではなく、オペランドリスト内の各オペランドのエラー値が返されます。これは、オペランドリストで重み値を調整する必要があるかどうかを判断するのに役立ちます。

オブジェクトリストは、TracePro での光学設計と最適化プロセスにおいて、オブジェクトの管理、スクリプトの実行、およびモデルの検証を行うための重要なインターフェースであると言えます。

Object list							
	Output?	ID	Object name	Object type	Mat. Catalog	Mat. Property	After-scheme
			Pre-processor				
	☒	0	Object 0	cExtrudedObject	None		
	☒	1	Object 1	cExtrudedObject	None		
	☒	2	Object 2	cDoubleSided...	None		

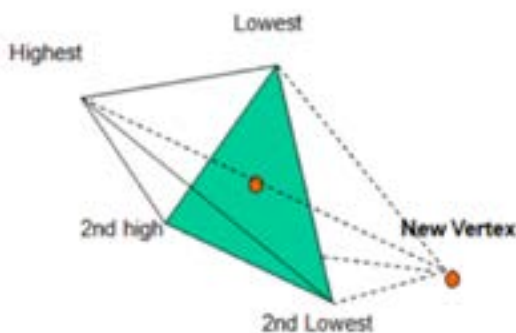
Output ? 出力 ?	TracePro にオブジェクトを送信するかどうかを決定するスイッチ。無効にした場合、アフタースキームも無視されます。
ID	オブジェクトの一意な ID 番号。
Object name オブジェクト名	オブジェクトの名前。
Object type オブジェクトタイプ	オブジェクトのタイプ。
Material catalog マテリアルカタログ	TracePro のプロパティデータベースから選択された、オブジェクトに適用される材料プロパティのカタログ。
Material property マテリアルプロパティ	TracePro のプロパティデータベースから選択された、オブジェクトに適用される材料プロパティ。
After scheme アフタースキーム	オプションが有効な場合、すべてのオブジェクトは関連付けられた後続のスキームコマンドを持つことができます。これはアフタースキームと呼ばれます。

最適化アルゴリズム

シンプレックス法（下り坂法）：

インタラクティブオプティマイザで使用されているアルゴリズムは、ネルダーとミード（Nelder and Mead）によるシンプレックス法（下り坂法）です。その説明とソースコードは、『Numerical Recipes in C++ 第2版』の第10.4章にあります。シンプレックス法（下り坂法）は、純粋に幾何学的手法を用いて設計を最適化します。初期のシンプレックスは、設計空間内を移動するために、反射、拡張、収縮、縮小を行います。

より低い誤差値（より良い性能）は、最も高い誤差値（最も悪い性能）の反対側に現れる可能性が高くなります。下の図は、ノードの反射の例を示しています。



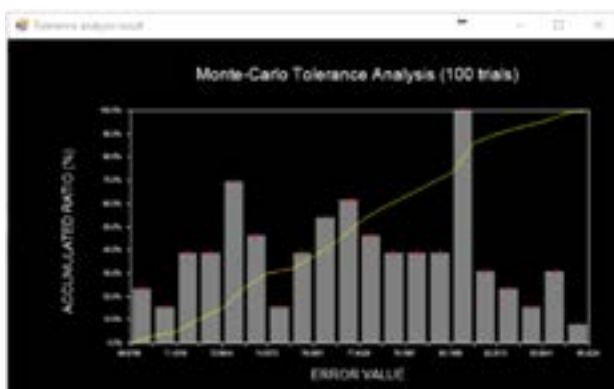
変数スキャン：

設計目標が最適なパラメータを見つけることではなく、より良い変数の範囲を見つけることである場合があります。例えば、適切な初期設計を決定する場合などです。この場合、変数スキャンは有用な手法です。各変数のステップと限界を定義する際に、各変数値のすべての組み合わせが結果のオブジェクトに適用されます。TracePro でシミュレーションと解析を実行し、保存されたデータとグラフを確認することで、最適な設計を簡単に見つけることができます。

公差解析

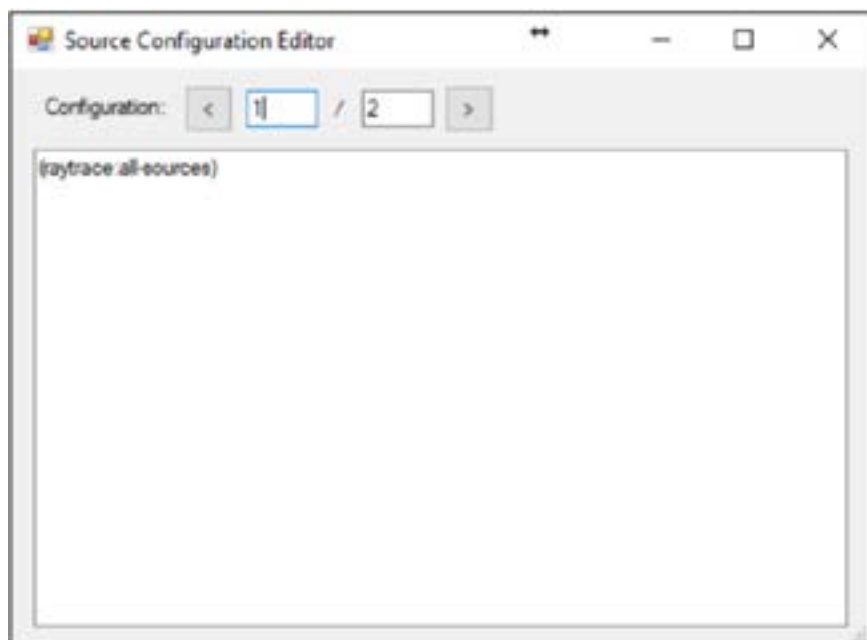
これはモンテカルロ法を用いた公差解析機能です。その考え方は、すべての変数を許容範囲内でランダムに変動させ、誤差値の分布に関する統計データを得ることです。各変数に対して選択できる摂動モードは3つあり、一様分布、正規分布（ガウス分布）、および端点です。一様分布では、最大値と最小値の間の任意の値を等しい確率で取得します。正規分布（ガウス分布）の場合、乱数は2シグマ点で打ち切られます。端点とは、変数の可能な生成値が最大値と最小値であることを意味します。

公差解析を開始する前に、試行回数を入力するための入力ボックスが表示されます。統計結果のリアルタイム更新は、公差解析ウィンドウに表示されます。



ソース構成

複数のソースを含むシステムの場合、システム全体の評価には、一部のソースを点灯したり、いくつかの光学部品を無効にしたりするなど、さまざまな構成が必要です。ソース構成は、単に光源を定義するだけでなく、オペランドが評価される前に実行できる一連のスキームのように機能します。ソース構成エディタでは、TracePro スキーム言語のみが受け入れられます。デフォルトのスキームマクロには (raytrace:all-sources) という 1 行しかありません。これは、TracePro に現在利用可能なすべての光源を使用してシミュレーションを実行するように命令するものです。



上記のように、TracePro のすべてのスキームコマンドは、ソースの構成で受け付けることができます。以下は、3 つの LED 光源すべての中から、赤い LED のみを点灯させる例です。

Scheme

```
(raytrace:set-surface-source-flag (entity:get-entity-by-name "LED_Red_Emitter") #t)
```

```
(raytrace:set-surface-source-flag (entity:get-entity-by-name "LED_Green_Emitter") #f)
```

```
(raytrace:set-surface-source-flag (entity:get-entity-by-name "LED_Blue_Emitter") #f)
```

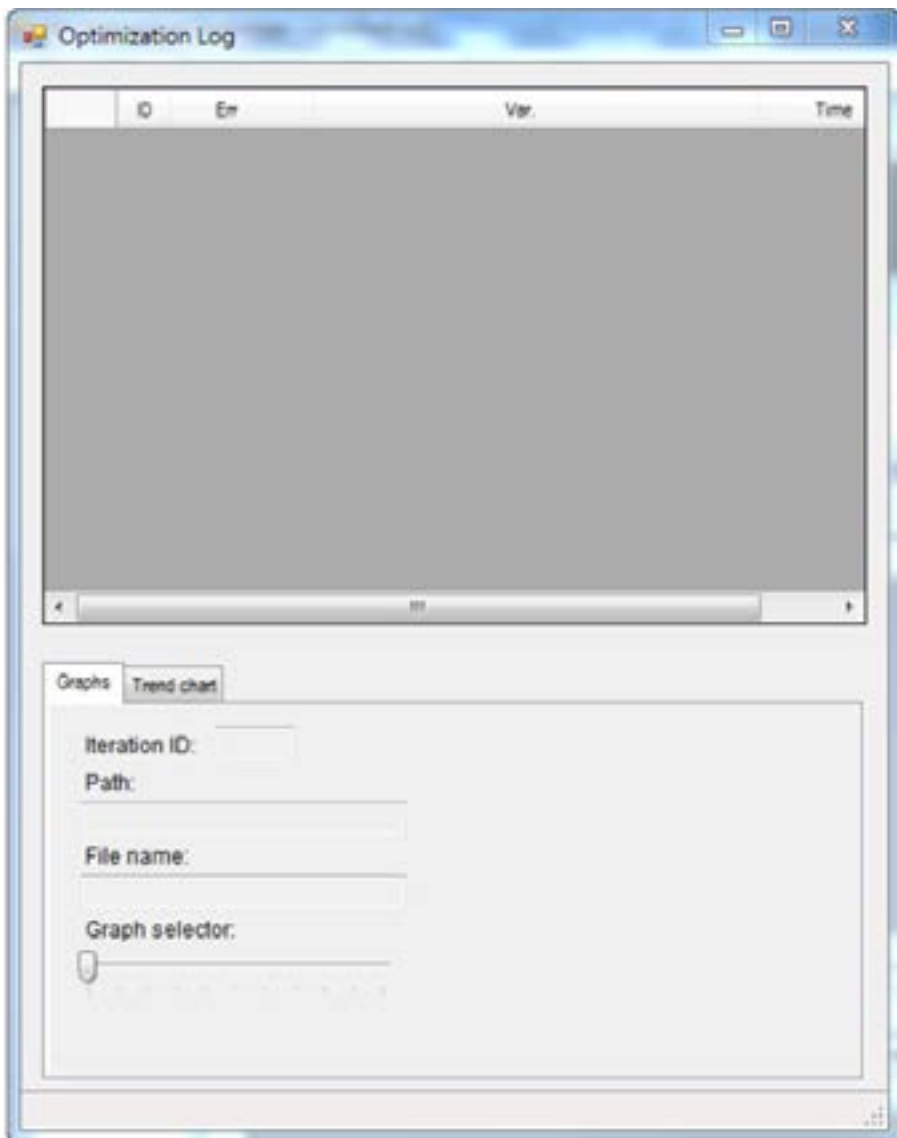
```
(raytrace:all-sources)
```

シミュレーションを開始するための実際のコマンドであるため、最後の行に (raytrace:all-sources) や (raytrace:file-source) などを追加することが重要です。

Add configuration 構成を追加	右側のテキストボックスの数値を増やすと、構成が追加されます。構成番号の変更を適用するには、Enter キーを押す必要があります。
Delete configuration 構成を削除	構成の総数を減らすと、末尾の構成が削除されます。

Optimization log（最適化ログ）

最適化ログは、最適化のすべての履歴を表示するためのウィンドウです。



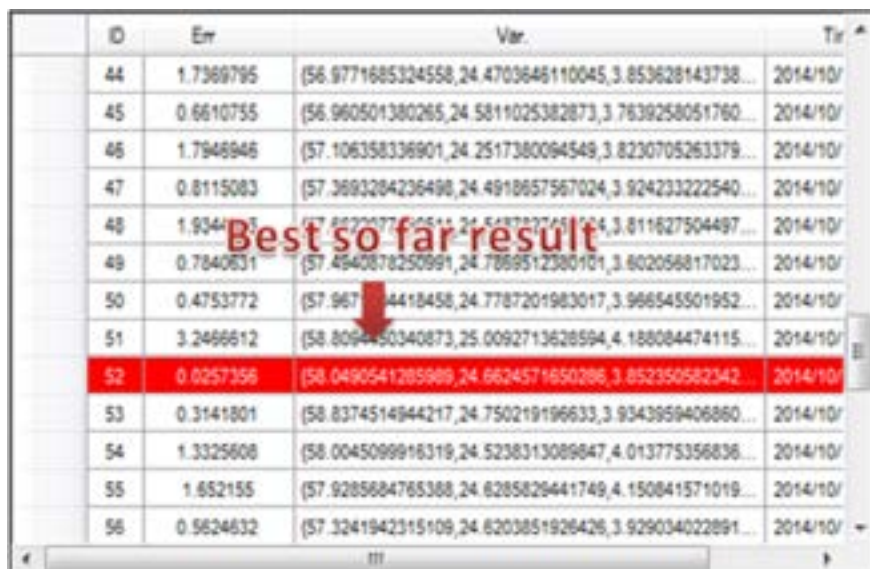
詳細

- [Optimization Log - User Interface 最適化ログ - ユーザーインターフェース](#)
- [Optimization Log - Update Model 最適化ログ - モデル更新](#)
- [Optimization Log - Log File 最適化ログ - ログファイル](#)

最適化ログ - ユーザーインターフェース

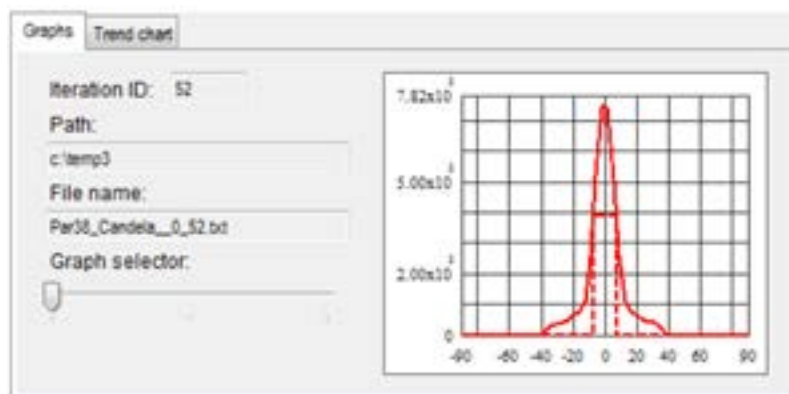
最適化ログは、最適化の全履歴を表示するためのウィンドウです。ログウィンドウの外観は以下の通りです。

上部のグリッドは、最適化の個々の試行を、そのシーケンス、エラー値、対応する変数、およびシミュレーション時間を一覧表示するためのものです。最も低いエラー値を持つイテレーションは赤色で強調表示されます。以前のパターンに戻すには、「Pattern name (パターン名)」コンボボックスを開き、パターンの名前をクリックします。プロファイルエディターウィンドウには、選択したパターンが表示されます。「赤いマイナス」ボタンをクリックすると、現在編集中的パターンが削除されます。

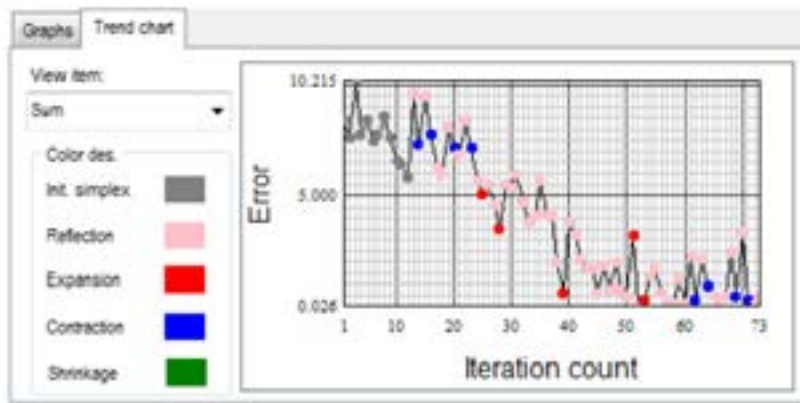


ID	Err	Var.	Tr
44	1.7369795	(56.9771685324558, 24.4703646110045, 3.853628143738...	2014/10/
45	0.6610755	(56.960501380265, 24.5811025382873, 3.7639258051760...	2014/10/
46	1.7946946	(57.106358336901, 24.2517380064549, 3.8230705263379...	2014/10/
47	0.8115083	(57.3693284236498, 24.4818657567024, 3.924233222540...	2014/10/
48	1.934...	(57.8527777777777, 24.5187777777777, 3.811627504497...	2014/10/
49	0.7840631	(57.4940878250991, 24.7869512380101, 3.802058817023...	2014/10/
50	0.4753772	(57.9677777777777, 24.4418458, 24.7787201983017, 3.966545501952...	2014/10/
51	3.2466612	(58.8094450340873, 25.0092713628594, 4.188084474115...	2014/10/
52	0.0257356	(58.0490541285989, 24.6624571650286, 3.852350582342...	2014/10/
53	0.3141801	(58.8374514944217, 24.750219196633, 3.9343959406880...	2014/10/
54	1.3325608	(58.0045099916319, 24.5238313089847, 4.013775356836...	2014/10/
55	1.652155	(57.9285684765388, 24.6285826441749, 4.150841571019...	2014/10/
56	0.5624632	(57.3241942315109, 24.6203851926426, 3.929034022891...	2014/10/

下部には、グリッドビューで選択されたイテレーションのグラフィック結果を確認するためのビューアが表示されます。「グラフ」タブには、選択されたイテレーションのイテレーション ID、保存されたパス、保存されたファイル名などの情報が表示されます。オペランドのタイプに応じて、グラフィックプロットは異なる形式で表示されます。「グラフセレクター」は、単一のイテレーションで生成された結果の中から選択するためのスライダーです。



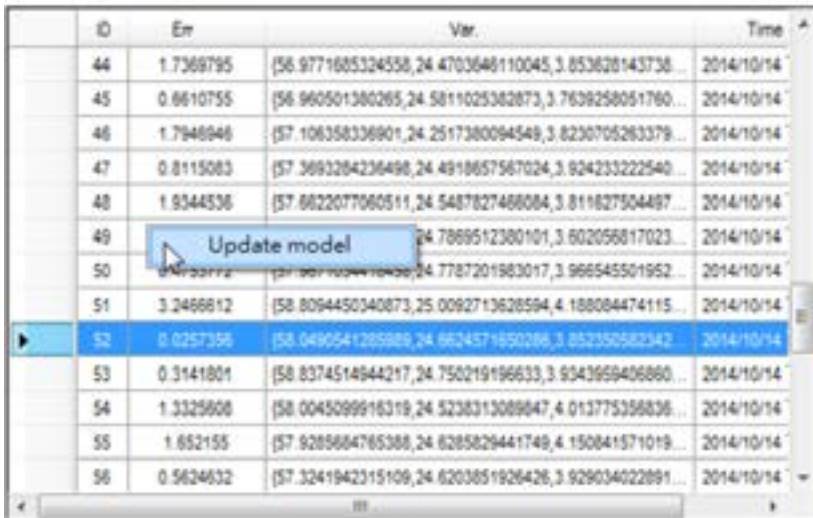
トレンドチャートタブは、最適化プロセス全体の履歴変化を表示します。インタラクティブオプティマイザで使用されている最適化アルゴリズムは、ダウヒル・シンプレックス法であり、最適化中にいくつかの幾何学的操作が行われます。トレンドチャートは、誤差値の変化だけでなく、各イテレーションで採用された幾何学的操作を異なる色で示します。



「View item」では、トレンドチャートに表示する値を選択します。デフォルトでは誤差の合計が表示されますが、各オペランドを個別に選択することも可能です。

最適化ログ - モデルの更新

グリッドビューの行を右クリックすると、ポップアップメニューが表示され、選択した行に属する変数値を使用してインタラクティブオプティマイザのモデルを更新します。



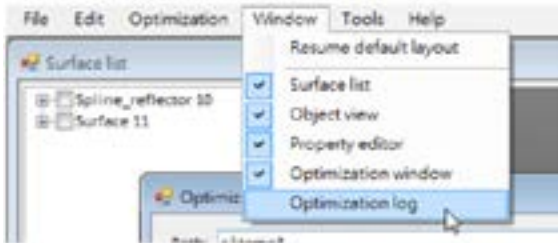
	ID	Err	Var.	Time
	44	1.7369795	(56.9771605324558, 24.4703648110045, 3.853620143738...	2014/10/14
	45	0.6610755	(56.960501380265, 24.5811025382873, 3.7638258061760...	2014/10/14
	46	1.7948946	(57.106358336901, 24.2517380094549, 3.8230705263379...	2014/10/14
	47	0.8115083	(57.3693284236498, 24.4918657567024, 3.924233222540...	2014/10/14
	48	1.9344536	(57.6822077060511, 24.5487827468084, 3.811827504487...	2014/10/14
	49	0.4152772	(54.7869512380101, 3.602056817023...	2014/10/14
	50	3.2466812	(58.8094450340873, 25.0092713628594, 4.188084474115...	2014/10/14
	51	0.0257356	(58.0490541285989, 24.6634571950286, 3.852350582342...	2014/10/14
	52	0.3141801	(58.8374514944217, 24.750219196633, 3.9343659406860...	2014/10/14
	53	1.3329608	(58.0045099916319, 24.5238313088647, 4.013775356836...	2014/10/14
	54	1.652155	(57.9285684765388, 24.6285829441749, 4.150841571019...	2014/10/14
	55	0.5624632	(57.3241942315109, 24.6203851926426, 3.929034022891...	2014/10/14

"Update model モデル更新" をクリックすると、選択したモデルを TracePro にも作成するかどうかを確認するプロンプトが表示されます。[はい] を選択すると、インタラクティブオプティマイザのモデルが更新され、TracePro のジオメトリも同様に更新されます。[いいえ] を選択すると、ユーティリティのモデルのみが更新され、TracePro には何も行われません。[キャンセル] は操作を停止します。

最適化ログ - ログファイル

最適化が実行されると、ログが同時に作成されます。履歴は、モデルファイルを再度保存する際にインタラクティブオブティマイザファイル (*.io2) に保存されます。

「ログファイルを復元するには、モデルファイルを開くだけで、すべてのログが読み込まれ、「ウィンドウ」->「最適化ログ」をクリックして開くことができる「最適化ログ」ウィンドウで表示できます。



Optimization operand(最適化オペランド)

インタラクティブオプティマイザは、以下のオペランドを提供します。

- Flux (フラックス): 光束。単位時間あたりにある面を通過する光のエネルギー量。
- CIE u'v' (CIE u'v' 色度): CIE 1976 UCS (均等色空間) の色度座標。色の違いを知覚的に均等に表現するのに用いられます。
- CIE xy (CIE xy 色度): CIE 1931 色度図の座標。色の特性を表す一般的な方法です。
- Irradiance (放射照度): 単位面積あたりに入射する放射束。表面が受け取る光の強さ。
- Irradiance profile (放射照度プロファイル): 特定の線や領域に沿った放射照度の分布。
- Intensity (光度): 特定の方向への単位立体角あたりの光束。点光源からの光の強さの方向依存性。
- Candela profile (カンデラプロファイル): 特定の角度範囲における光度の分布。照明器具などの配光特性を示します。
- Uniformity (均一度): 表面における照度や輝度の均一性の度合い。
- Beam width (ビーム幅): 光ビームの広がりを示す指標。
- Spot size (スポットサイズ): 光が照射された際のスポットの大きさ。
- Beam center (ビーム中心): 光ビームの中心位置。
- Spot size (スポットサイズ): (上記と重複していますが、文脈によっては異なる意味合いを持つ可能性があります。)
- Luminance (輝度): 単位面積あたり、単位立体角あたりの光度。観察者が見る表面の明るさ。
- Scheme (スキーム): TracePro のスクリプト言語であるスキーム言語のコマンドを実行し、その結果を最適化の評価基準として使用します。
- Photorealistic (フォトリアリスティック): レンダリングされた画像の特定のピクセルの色や輝度などの情報を最適化の評価基準として使用します。
- User-defined operand (ユーザー定義オペランド): ユーザーが独自に定義する最適化の評価基準。スキーム言語などを用いてカスタムの計算や評価を行うことができます。

オペランドタイプ (Operand type)	Opt. (演算)	サーフェス (Surface)	範囲 (Range)	重み (Weight)	目標値 (Target value)	マップ演算 (Map Op.)	ソース構成 (Src Config.)
フラックス (Flux)	合計 (Sum) 正規化 (Normalized)	< 照射照度 マップのサー フェス名 > (Surface name of irradiance map)	全マップ (Whole map) 領域 (Region)	0 より 大きい (greater than 0)	目標フラックス 値 (Target flux value)	照射照度 / 照度マッ プオプション (Irradiance/ Illuminance Map Options)	構成数 (number of configs)
CIE u'v'	平均 (Average)	< 照射照度 マップのサー フェス名 > (Surface name of irradiance map)	全マップ (Whole map) 領域 (Region) 点 (Point)	0 より 大きい (greater than 0)	目標色度座標 (Target color coordinate)	照射照度 / 照度マッ プオプション (Irradiance/ Illuminance Map Options)	構成数 (number of configs)
CIE xy	平均 (Average)	< 照射照度マッ プのサーフェス 名 > (Surface name of irradiance map)	全マップ (Whole map) 領域 (Region) 点 (Point)	0 より 大きい (greater than 0)	目標色度座標 (Target color coordinate)	照射照度 / 照度マッ プオプション (Irradiance/ Illuminance Map Options)	構成数 (number of configs)

照射照度 (Irradiance)	平均 (Average) 最大 (Max) 最小 (Min)	< 照射照度 マップのサー フェス名 > (Surface name of irradiance map)	全マップ (Whole map) 領域 (Region) 点 (Point)	0 より大 きい (greater than 0)	目標照射照度値 (Target irradiance value)	照射照度 / 照度マッ プオプション (Irradiance/ Illuminance Map Options)	構成数 (number of configs)
照射照度 プロファイル (Irradiance profile)	類似度 (Similarity)	< 照射照度マッ プのサーフェ ス名 >(Surface name of irradiance map)	< 使用なし > (No use)	0 より 大きい (greater than 0)	< 目標プロファ イルを定義 > (Define the target profile)	照射照度 / 照度マッ プオプション (Irradiance/ Illuminance Map Options)	構成数 (number of configs)
光度 (Intensity)	平均 (Average) 最大 (Max) 最小 (Min) < 空欄はミス し た光線を使用 > (empty is using missed ray)	< カンデラプ ロットのサー フェス名 > (Surface name of Candela plot)	全マップ (Whole map) 領域 (Region) 点 (Point)	0 より 大きい (greater than 0)	目標光度値 (Target intensity value)	カンデラオプション (Candela Options)	構成数 (number of configs)
カンデラ プロファイル (Candela profile)	類似度 (Similarity) < 空欄はミス し た光線を使用 > (empty is using missed ray)	< カンデラプ ロットのサー フェス名 > (Surface name of Candela plot) ミスした光線 (Missed ray) 出射光線 (Exiting ray) 入射光線 (Incident ray)	< カンデラプロッ トのサーフェス名 > (Surface name of Candela plot) ミスした光線 (Missed ray) 出射光線 (Exiting ray) 入射光線 (Incident ray)	0 より 大きい (greater than 0)	< 目標プロファ イルを定義 > (Define the target profile)	カンデラオプション (Candela Options)	構成数 (number of configs)
均一度 (Uniformity)	プログラム (Programmed)	< 照射照度 マップのサー フェス名 > (Surface name of irradiance map)	< 使用なし > (No use)	0 より 大きい (greater than 0)	< 均一度目標を 定義 > (Define uniformity target)	照射照度 / 照度マッ プオプション (Irradiance/ Illuminance Map Options)	構成数 (number of configs)
ビーム幅 (Beam width)	FWHM	< 空欄はミスし た光線を使用 > (empty is using missed ray)	< カンデラプロッ トのサーフェス名 > (Surface name of Candela plot) ミスした光線 (Missed ray) 出射光線 (Exiting ray) 入射光線 (Incident ray)	0 より 大きい (greater than 0)	目標ビーム幅 (Target beam width)	カンデラオプション (Candela Options)	構成数 (number of configs)
スポットサイズ (Spot size)	デフォルト (Default)	< 照射照度 マップのサー フェス名 > (Surface name of irradiance map)	< 使用なし > (No use)	0 より 大きい (greater than 0)	目標スポット サイズ (Target spot size)	照射照度 / 照度マッ プオプション (Irradiance/ Illuminance Map Options)	構成数 (number of configs)

輝度 (Luminance)	合計 (Sum) 平均 (Average) 最大 (Max) 最小 (Min) RMS コントラスト (Contrast)	< 使用なし > (No use)	< 輝度を設定 > (Setup luminance)	0 より 大きい (greater than 0)	目標値 (Target value)	使用なし (No use)	構成数 (number of configs)
スキーム (Scheme)	プログラム (Programmed)	< 使用なし > (No use)	< 使用なし > (No use)	0 より 大きい (greater than 0)	プログラミングコード (Programming code)	使用なし (No use)	構成数 (number of configs)
フォトリアリスティック (Photorealistic)	平均 (Average) 最大 (Max) 最小 (Min) RMS	< 使用なし > (No use)	全マップ (Whole map) 領域 (Region) 点 (Point)	0 より 大 き い (greater than 0)	目 標 輝 度 値 (Target luminance value)	フォトリアリスティックレンダリング設定 (Photorealistic rendering setup)	構成数 (number of configs)

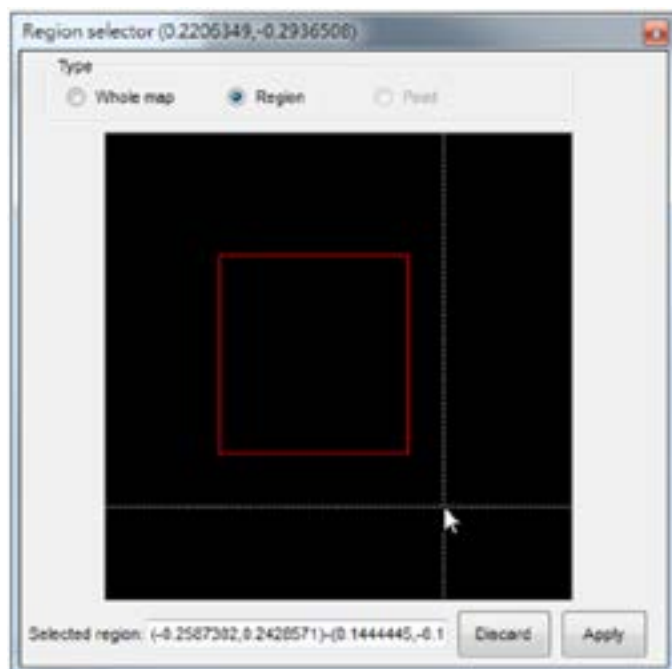
詳細：

- **Flux: 光束 (こうそく)** - 光の明るさの量。単位はルーメン (lm)。
- **CIE u'v': CIE u'v' 色度図** - 色を表すための国際照明委員会 (CIE) が定めた色空間の一つ。知覚的に均等な色空間に近い。
- **CIE xy: CIE xy 色度図** - 色を表すための国際照明委員会 (CIE) が定めた色空間の一つ。色の三刺激値から計算される。
- **Irradiance: 放射照度** - 単位面積あたりに入射する放射エネルギーの量。単位はワット毎平方メートル (W/m²)。照明分野では照度 (ルクス、lx) と混同される場合があるが、厳密には異なる。
- **Irradiance profile: 放射照度分布** - 面上における放射照度の変化を示すグラフや図。
- **Intensity: 光度** - 特定の方向への光の強さ。単位はカンデラ (cd)。
- **Candela profile: カンデラ分布** - 光源から測定方向に放射される光度の強さを示すグラフや図。
- **Uniformity: 均一度** - 面上における光の明るさや色の均一性の度合い。
- **Beam width: ビーム幅** - 光線の広がりを示す角度。通常、光度が最大値の半分になる角度で定義される。
- **Spot size: スポット径** - 光線が照射する面の明るい領域の大きさ。
- **Luminance: 輝度** - 光源または反射面から特定の方向へ放射される光の明るさの度合い。単位はカンデラ毎平方メートル (cd/m²)。
- **Scheme: スキーム** - 計画、構成、体系。ソフトウェアの文脈では、特定の機能や処理を行うための手順や設定を指すことがある。
- **Photorealistic: フォトリャリスティック** - 写真のようにリアルな、実写のような。
- **User-defined operand: ユーザー定義オペランド** - ソフトウェアの最適化機能などで、ユーザーが独自に定義する評価項目や目標値。

Flux フラックス

フラックスターゲットは、指定された領域にわたる照度マップを積分することによって決定されます。

領域の選択肢として「マップ全体」または「領域」を選択できます。「マップ全体」が選択された場合、インタラクティブオプティマイザは、マップ全体でフラックスを再計算する代わりに、照度マップからの合計フラックス値をそのまま使用します。「領域」が選択された場合、選択された領域内のフラックスは、照度マップを分析することによって計算されます。

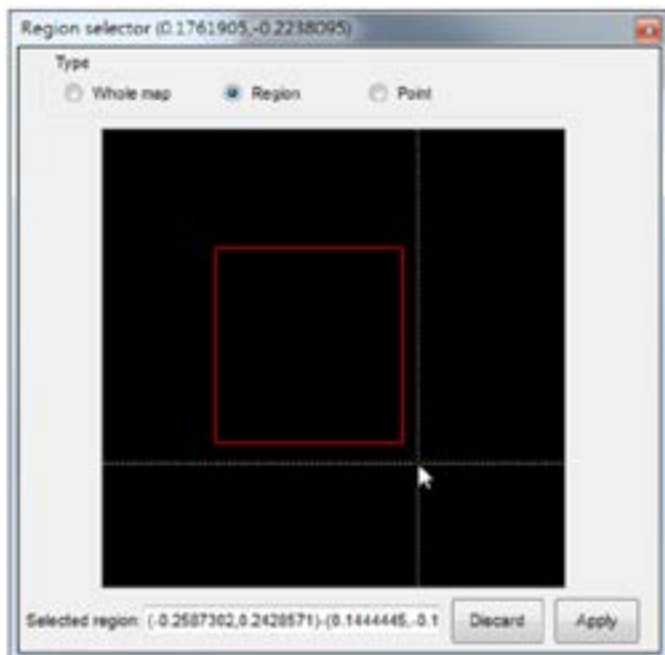


フラックスオペランドの目標値は、その望ましい値です。したがって、誤差の値は、計算された値から目標値を引いたものの絶対値となります。

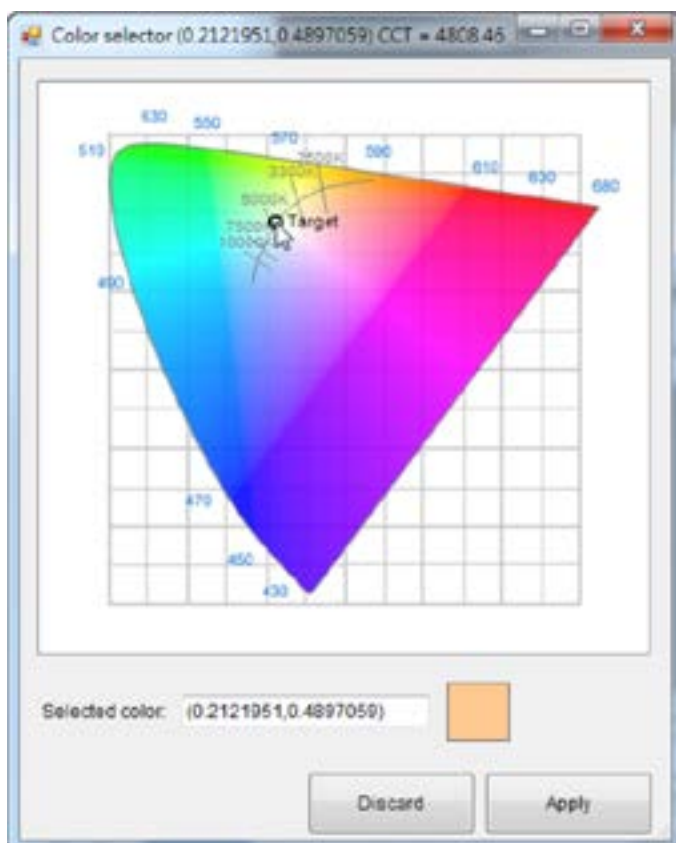
CIE u'v' および CIE xy

最適化の目標が色である場合、使用している色度空間に応じて2つのオプションがあります。

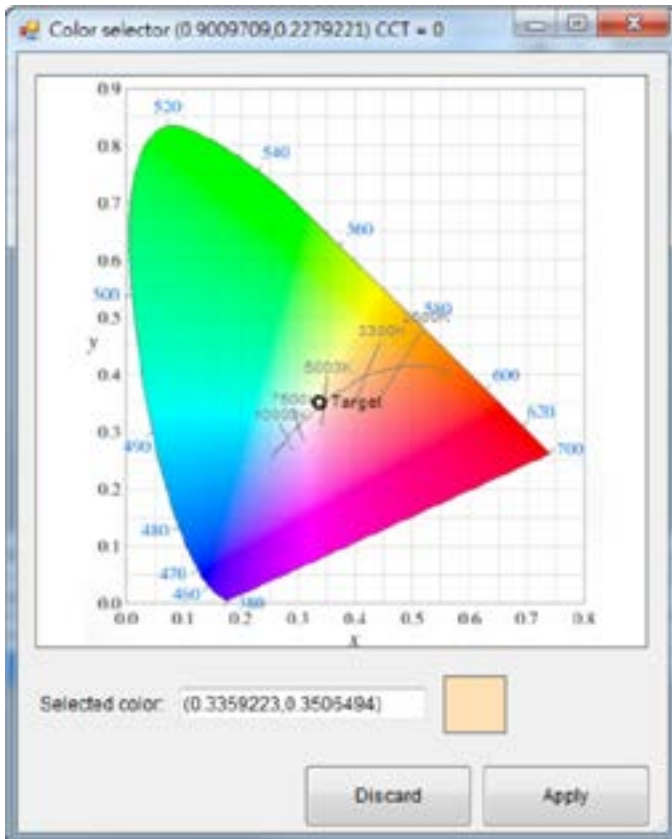
CIE u'v' 色空間（CIE 1976 UCS 図としても知られています）は、TracePro で利用可能なカラーマップの1つです。CIE u'v' オペランドは、ターゲットの定義方法として、Whole map（全体マップ）、Region（領域）、Point（点）の各メソッドをサポートしています。Point メソッドは指定された位置の色値を取得し、Whole map メソッドはマップ全体の平均値を計算し、Region メソッドは指定された領域の平均値を計算します。



得られたカラー値は色座標における位置であり、誤差値は色度図上における現在の座標と目標座標との間の距離です。例えば、計算された色（選択された領域内で平均化されたもの）が (0.2, 0.2) であり、目標の色座標が (0.22, 0.5) である場合、誤差値は $\sqrt{(0.22-0.2)^2+(0.5-0.2)^2} = 0.3$ となります。



CIExy 色空間は、CIE 1931 色度図のことです。このオペランドは、CIE u'v' オペランドと同じ方法で定義され、計算されます。

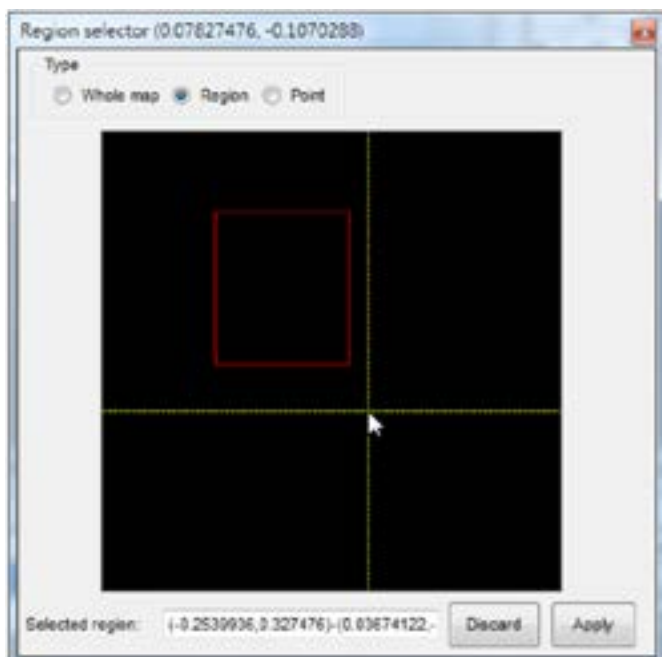


Irradiance 放射照度

放射照度ターゲットは、指定された表面上の特定の領域における放射照度マップからデータを取得します。

「範囲」のセルをクリックすると、データ取得範囲を指定するためのダイアログが表示されます。データ取得領域として、マップ全体、範囲、または単一点を選択できます。放射照度はフラックスのような積分値ではなく単一の値であるため、「Opt」セルの選択が重要です。「Opt」セルでは、平均、最大、最小の3つのオプションを選択できます。

平均は、指定された領域全体の平均値を取得します。最大は、選択された領域内の最高の放射照度を取得します。最小は、選択された領域内の最低値を取得します。



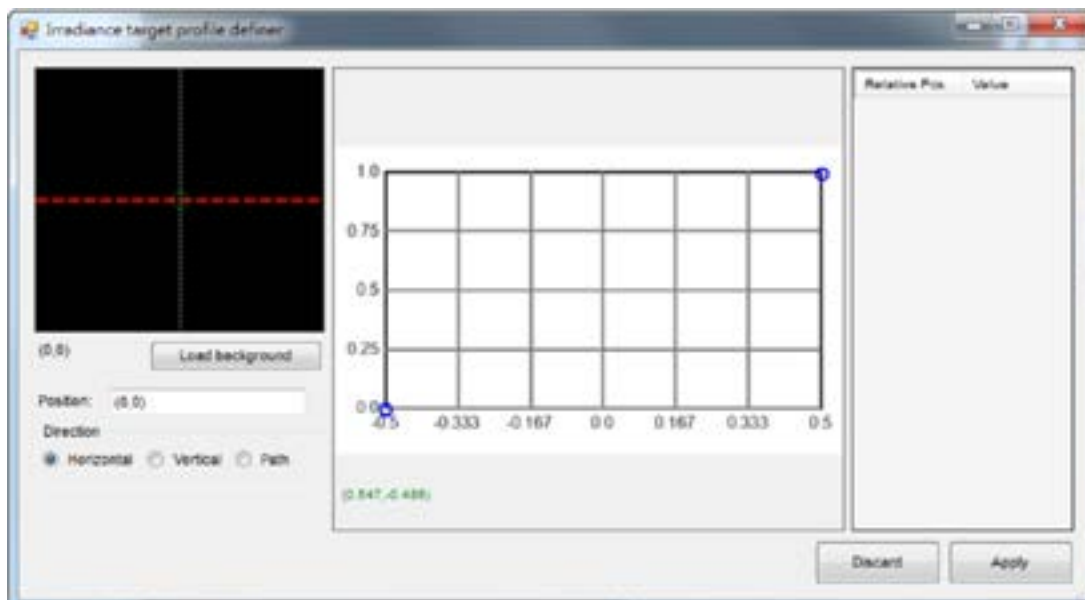
照度オペランドの目標値は、その望ましい値です。したがって、誤差値は、計算された値から目標値を引いたものの絶対値となります。

Irradiance profile 放射照度プロファイル

放射照度プロファイルオペランドは、放射照度マップからプロファイルとして取得されます。計算されたプロファイルと目標プロファイルを比較することで、誤差関数を作成できます。

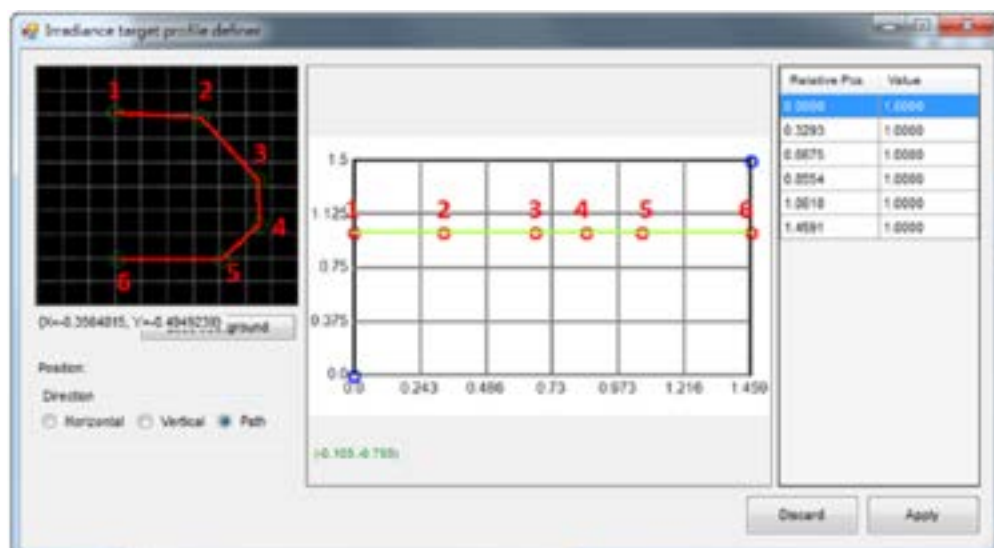
使用される比較方法は、ピアソン相関です。ここで計算された係数は、最適化のための正の値の要件により、0～2の範囲にシフトされていることに注意してください。

類似性オペランドでは範囲を定義する必要がないため、「範囲」セルは非アクティブになっています。必要なすべてのパラメータは、「目標」セルをクリックして起動されるダイアログボックスに入力します。



照度ターゲットプロファイル定義ツールで最も重要なオプションは「方向？」であり、定義するターゲットプロファイルを決定します。

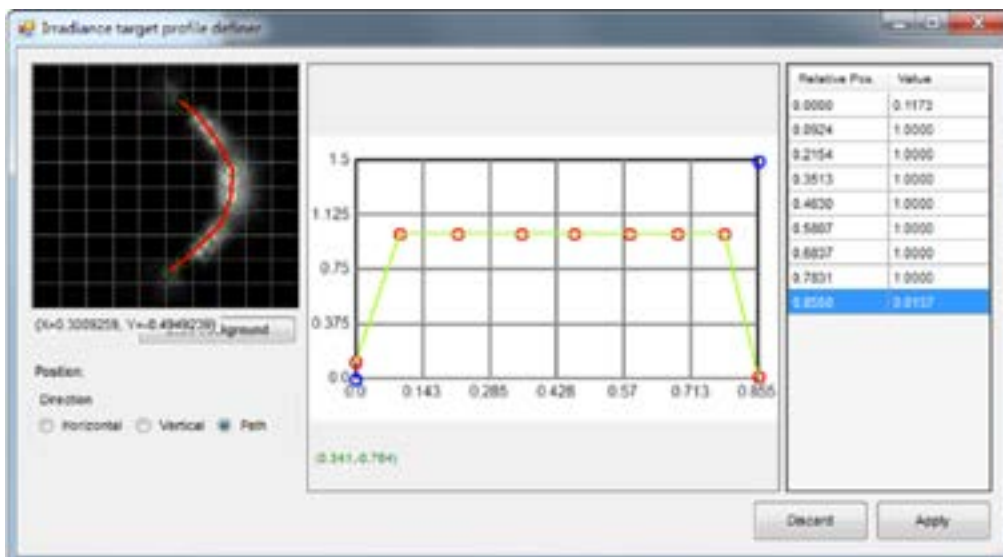
水平 (Horizontal)：	プロファイルは水平な経路に沿って作成されます。その位置は、ターゲットマップ上でのマウスのクリックによって指定するか、テキストボックスに手動で入力します。
垂直 (Vertical)：	プロファイルは垂直な経路に沿って作成されます。その位置は、ターゲットマップ上でのマウスのクリックによって指定するか、テキストボックスに手動で入力します。
パス (Path)：	プロファイルは単純な直線ではなく、カスタマイズされた経路に沿って作成されます。「パス？」を選択すると、ターゲットプロファイル定義ツールは特別なモードに切り替わり、ターゲットマップをクリックすることで複数の点を入力できます。



挿入された点は、点を右クリックし、ポップアップメニューから「削除」を選択することで削除できます。点の順序は変更できないため、プロファイルパスを調整する際にはこの順序を維持する必要があることに注意してください。

カスタムパスプロファイルの適用は、空のマップ上では定義が難しい場合があります。そのため、「背景をロード」ボタンをクリックし、以前に保存した照度マップの場所を参照することで、既存の照度マップを背景として追加できます。この背景は、パスプロファイルを定義するためのテンプレートとして使用できます。既存の照度上にプロファイルパスを描画することは、最適化プロセス中にオペランドが正しく機能することを確認する上で役立ちます。

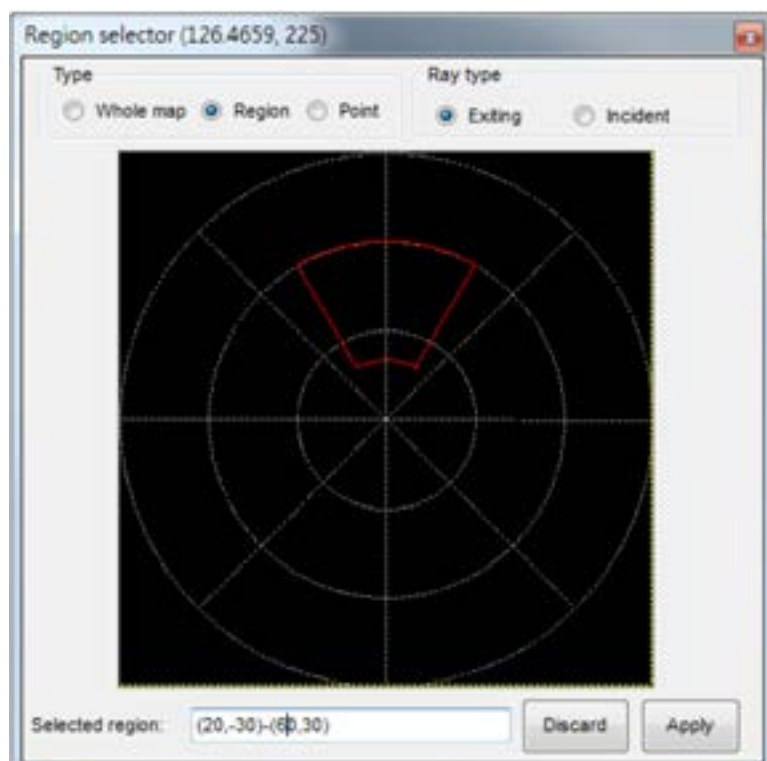
中央のプロットは、定義されたパスに沿った点の相対照度プロファイルを定義します。カスタマイズされたパスに沿った点は、パスを平面断面に拡張することによってプロットされます。照度プロットの x 値は、隣接する 2 点間の長さによって計算されます。



Intensity 光度

光度ターゲットは、特定の領域にわたるシミュレートされた極座標カンデラプロットからデータを取得します。

同様のオペランドである照度と同様に、光度オペランドはそのサンプリング範囲を「マップ全体」、「領域」、および「点」の3つのオプションから選択できます。「Opt.」セルでの選択は、選択された範囲内のデータに適用される演算子を決定します。「平均」は、選択された領域内のすべてのデータの平均値を取得します。「最大」は、最大の光度を取得し、「最小」は、最小の光度値をそのオペランドとして取得します。



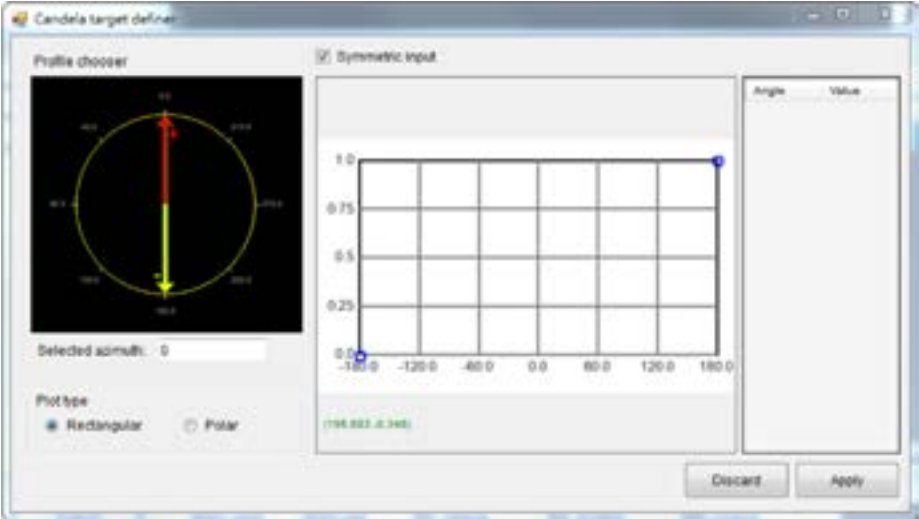
このオペランドの解析対象として、出射光線または入射光線のどちらを使用するかを選択するのが「光線タイプ」です。なお、サーフェスセルが空のままの場合、「出射」または「入射」のどちらを選択しても、未着光線を使用して解析することを示します。

強度オペランドの目標値は、その望ましい値です。したがって、誤差値は、計算値の絶対値から目標値を引いた値となります。

Candela profile カンデラプロフィール

カンデラプロフィールオペランドは、カンデラマップからプロフィールとして取得されます。計算されたプロフィールと目標プロフィールを比較することで、誤差関数を作成できます。

目標セルをクリックすると、カンデラ目標プロフィール定義ツールが開きます。ウィンドウのレイアウトは照度プロフィール定義ツールと似ていますが、ウィンドウの左側はマップではなくプロフィール選択ツールです。



プロフィール選択ツールの操作は簡単です。ユーザーはプロフィール選択ツール上でマウスをクリックするか、選択した方位角ボックスに希望の方位角を入力することで、任意の方位角スライスを選択できます。

Plot type プロットタイプ	A（矩形）または極座標プロットタイプを選択できます。これにより、目標プロフィールを入力する際に使用される描画タイプが変更されます。この設定は、保存されるカンデラマップのタイプにも影響します。
Symmetric input 対称入力	このボックスにチェックを入れると、プロフィールウィンドウに入力された任意の制御点が、中心線の反対側に反映またはミラーリングされます。

Uniformity 均一度

場合によっては、要求される性能がフラックス値や単純なプロファイルでは記述できないことがあります。特定の領域におけるより広範な性能を表すためには、より複雑なオペランドが必要となる場合があります。「均一度」オペランドは、カスタマイズされた性能目標を評価するためのより柔軟な方法を提供します。

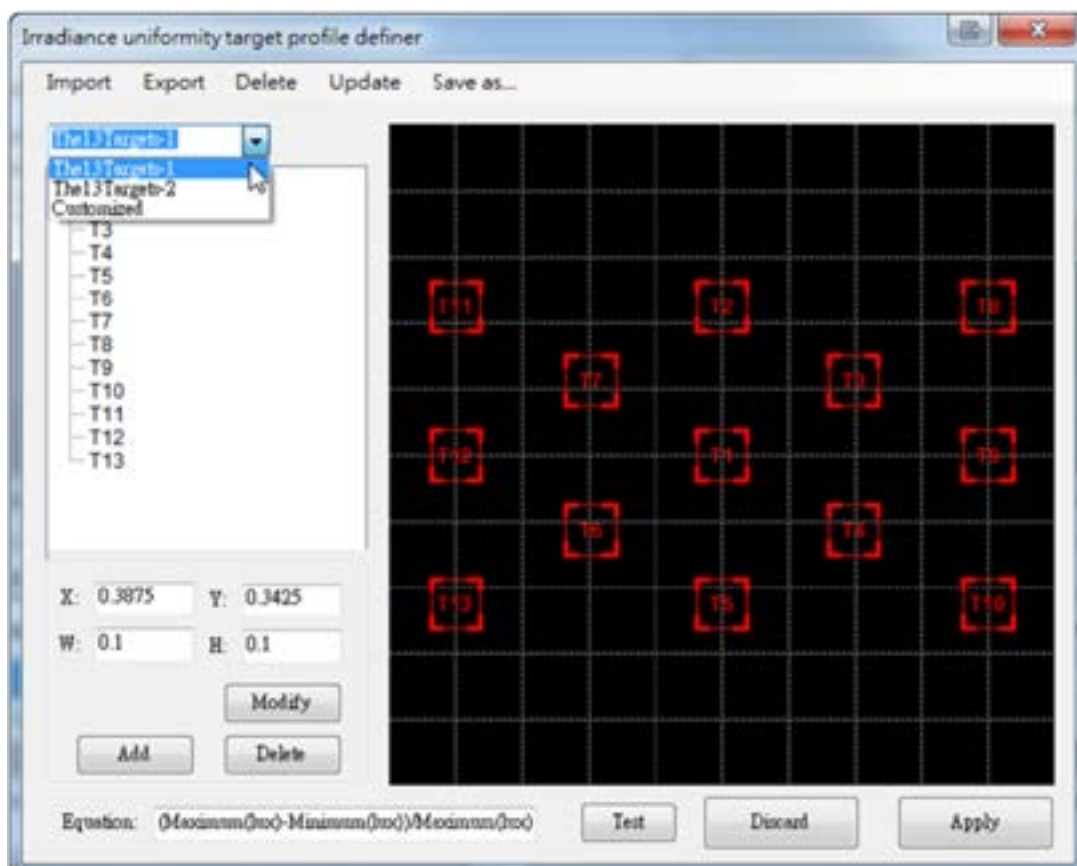
「目標値」セルをクリックすると、「照度均一度目標プロファイル定義」が開きます。「均一度」オペランドは、「照度均一度目標定義」の「計算方法?」コンボボックスで選択できる 2 つの組み込みデフォルトメソッドを提供します。カスタムメソッドは、オペランドダイアログボックスで作成することもできます。すべてのカスタムメソッドはデータベースに保存されるため、将来再利用できます。すべての均一度プリセットを保存するファイルは、…\AppData\Roaming\Lambda Research Corporation\3D Interactive Optimizer\UData.txt にあります。

均一度オペランドを追加する最も簡単な方法は、コンボボックスで「The13Target-1?」などのデフォルトメソッドを選択することです。「The13Target-1?」プリセットには、照度マップ上の事前定義された位置とサイズに 13 個のサンプリングターゲットがあります。プリセットメソッドを選択すると、「数式?」テキストボックスに関連する計算式が表示されます。

結果の誤差を計算するために、いくつかの関数が提供されています。「The13Target-1?」プリセットを例として、オペランドの仕組みの説明を以下に示します。「The13Target-1?」メソッドの事前定義された数式は「(最大(ルクス) - 最小(ルクス)) / 最大(ルクス)」です。ここで：

- 最大(ルクス): 13 個のすべてのターゲット領域の中で最大の照度値を取得します
- 最小(ルクス): 13 個のすべてのターゲット領域の中で最小の照度値を取得します

この計算方法の結果の値は、 $(Irradiancemax - Irradiancemin) / Irradiancemax$ {(照度最大 - 照度最小) / 照度最大} であり、これは特定の領域における均一度の一般的な定義です。



カスタマイズされた計算方法を作成するには、「Calculation Methods 計算方法」コンボボックスで「Customized? カスタマイズ?」を選択します。次に、「Add? 追加」ボタンをクリックして、右側の照度マップに長方形を作成します。範囲は分数で、x 軸と y 軸の値は -0.5 から +0.5 であることに注意してください。新しく生成された長方形は、マップ上の任意の位置にドラッグできます。長方形のいずれかの角にカーソルを合わせると、カーソルの形状が変わり、角をドラッグしてターゲットのサイズを変更できます。より正確なターゲットの位置とサイズについては、適切なボックスに目的の値を入力し、「Modify 修正」をクリックします。ターゲットを削除するには、ターゲットを選択してから、「Delete? 削除」ボタンをクリックします。

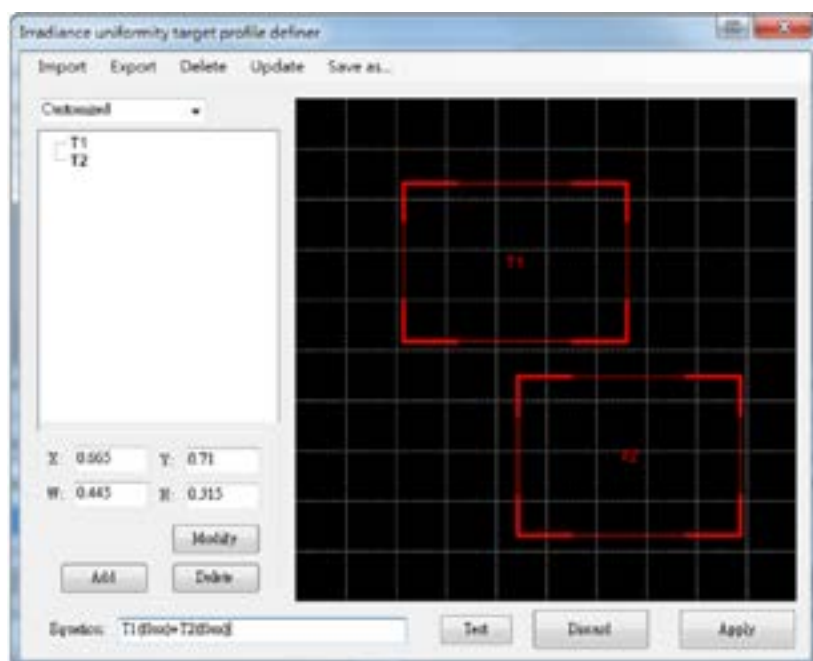
照度マップ上のサンプリング領域ターゲットが定義されたら、計算式を定義する必要があります。これは、オペランドの結果の値を計算するために使用される式です。前の段落のデフォルトの「The13Target-1」メソッドの説明では、グローバルに定義された関数である Maximum(lux) と Minimum(lux) を使用して、定義されたすべてのターゲットから極値を取得すると述べました。これらの 2 つのグローバル関数に加えて、Average は均一度オペランドにおけるもう 1 つのグローバルに定義された関数です。

Maximum、Minimum、および Average の後のキーワード「lux」は、極値または平均を検索する際に評価する値を指定するために使用されます。

「lux」に加えて、均一度オペランドは値に対して次のオプションも提供します。

lux	- 各ターゲット矩形内の平均照度
flux	- 各ターゲット矩形内の全光束
area	- 各ターゲット矩形の面積
centerx	- 各ターゲット矩形の中心位置の x 座標成分
centery	- 各ターゲット矩形の中心位置の y 座標成分
width	- 各ターゲット矩形の幅
height	- 各ターゲット矩形の高さ
uniformity	- 各ターゲット矩形内の均一度指数を、「The13Target-2?default」メソッドの定義を用いて計算します。

特定のターゲットの独立した値は、ターゲット ID を使用して取得できます。たとえば、T1 と T2 という名前の 2 つのターゲットの全光束の合計を見つけるには、方程式は「T1(flux)+T2(flux)?」となります。



ルクスやフラックスのような特定の値を返すためにキーワードと組み合わせて使用されるローカル関数とグローバル関数は、与えられた光学システムの性能を評価および最適化するためのカスタムメソッドを作成する上で、非常に大きな柔軟性を提供します。これらのカスタムメソッドは、保存、削除だけでなく、他のユーザーが使用するためにエクスポートおよびインポートすることも可能です。

ビーム幅

ビーム幅は、指定されたカンデラプロットの半値全幅（FWHM）を評価するためのものです。カンデラプロファイルオペランドと同様に、ビーム幅オペランドにも「Surface 表面」と「Range 範囲」のオプションがあります。

「Surface 表面」は、出射光線と入射光線のカンデラプロットを収集する表面名を指定するためのものです。迷光線がオペランドのターゲットとして必要な場合、「Missed ray 迷光線」を選択するか、「Surface」のセルを空のままにすることで、同じ結果が得られます。

Beam center ビームセンタ

Beam center ビームセンタは、指定されたカンデラプロットのピーク最大値を評価するためのものです。カンデラプロファイルオペランドと同様に、ビーム中心オペランドにも「Surface」と「Range 範囲」のオプションがあります。

「Surface」は、出射光線と入射光線のカンデラプロットを収集する表面名を指定するためのものです。もし迷光線をオペランドのターゲットとして必要とする場合、「Missed rays」を選択するか、「Surface」のセルを空のままにすることで、同じ結果が得られます。

Spot size スポットサイズ

スポットサイズとは、指定されたサーフェス上で、輸送フラックスで重み付けされた光線の RMS スポットサイズを計算するものです。RMS スポットサイズの出力は、スポットの半径（半直径）を指します。誤差は、計算された RMS スポットサイズから目標値を引いたものです。

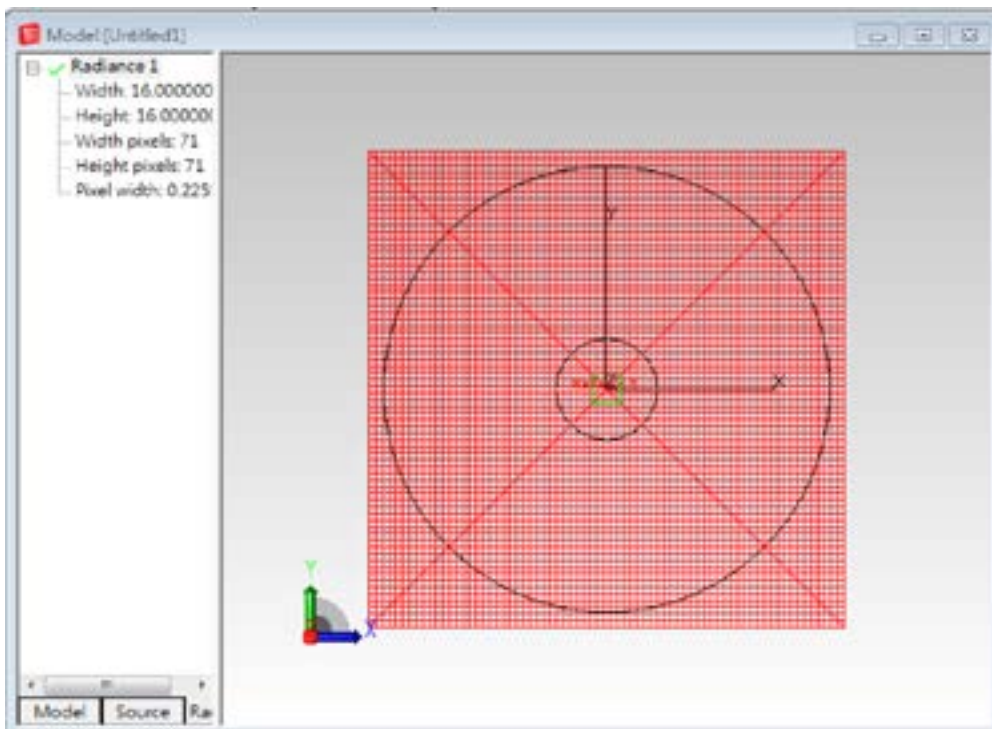
Luminance 輝度

輝度は TracePro で行うことができる、より特殊な解析です。特定の目の位置と視野角における放射輝度 / 輝度値を評価するために使用されます。インタラクティブオプティマイザは、輝度マップを最適化目標の 1 つとして扱うことができるようになりました。

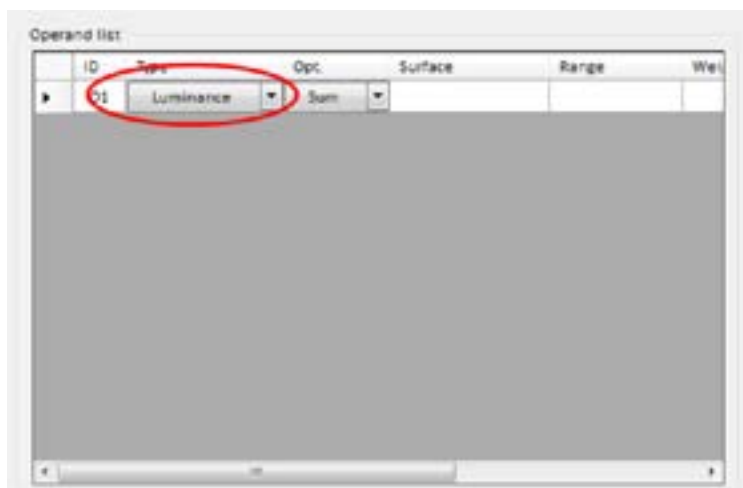
これを使用するには、まず TracePro で動作可能な輝度解析を設定し、Range のセルから起動したダイアログに入り、解析に使用するデータの範囲を設定する必要があります。

以下は、正しい輝度オペランドを設定する方法を示す例です。

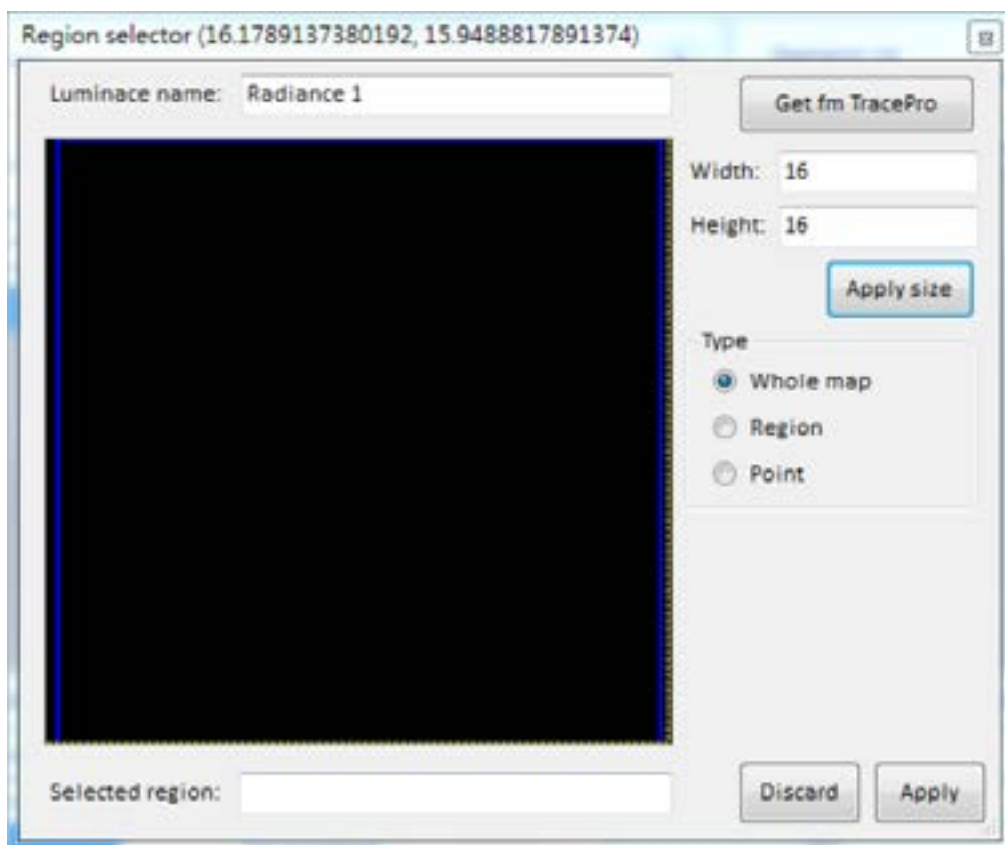
1. TracePro で輝度解析を作成します。ここでは、その名前を「Radiance 1」とします。



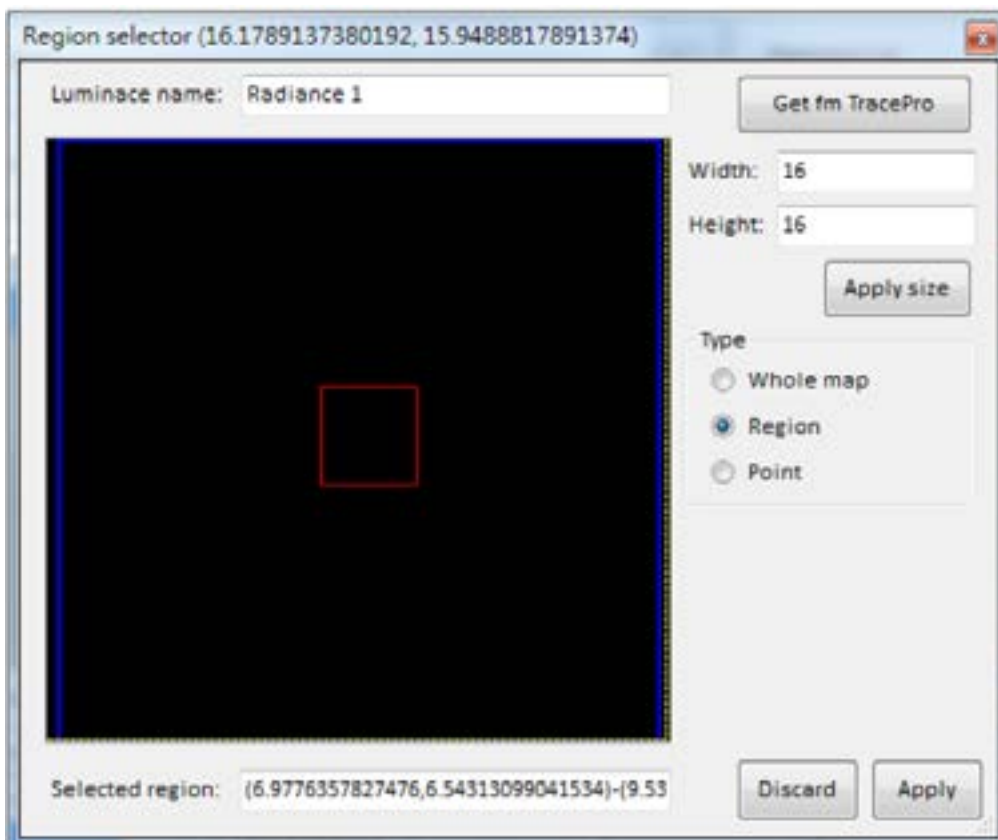
2. オペランドとして「輝度」を選択します。



3. オペランドをさらに設定するために、「Range 範囲」のセルをクリックしてください。テキストボックスに事前に作成した放射輝度解析の名前を入力し、「Get fm TracePro」というタイトルのボタンを押して、放射輝度マップのサイズを取得し、取得したサイズを埋めてください。



4. 範囲の種類を「Whole map 全マップ」、「Region 領域」、「Point 点」の中から選択してください。ここでは範囲の種類として「Region 領域」を選択し、解析する指定範囲としてエディタ上で矩形をドラッグします。「Apply 適用」を押して設定を確定します。



5. シミュレーションデータに適用したいオプションを選択してください。注記：Sum オプションを選択すると、選択した領域内の全フラックス（光束）が計算されます。Average オプションは平均化された放射輝度を取得し、Max、Min、および Contrast オプションはシミュレーション結果で報告された放射輝度の値に基づいています。

User defined ユーザー定義

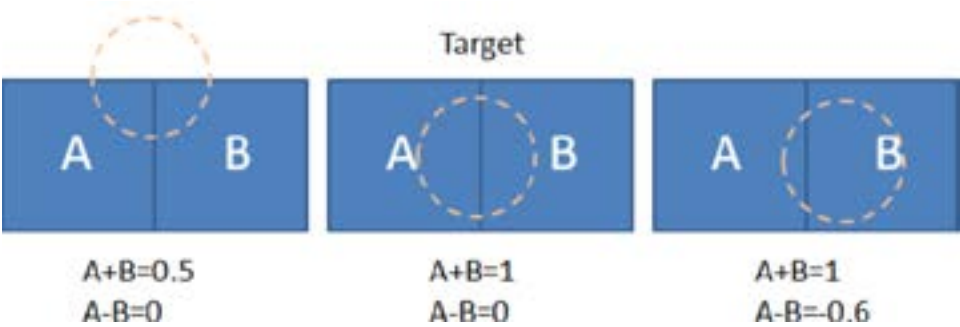
ユーザー定義オペランドは、複数の既存のオペランドを数式を用いて組み合わせ、新しい量を定義する機能を提供します。オペランド ID は、既存のオペランドをユーザー定義オペランドに追加するために使用されます。

各オペランドは、0+ 番号の形式の一意の ID を持ちます。オペランドの ID が数式で使用されると、そのオペランドの値が計算に使用されます。

例えば、設計目標が特定の位置での照度を持つことである場合、DetectorA と DetectorB という名前の 2 つの隣接する検出器、またはターゲットを作成することから始めることができます。



もし設計目標が二つのターゲット間で照度を均一にすることであるならば、結果は下の真ん中の図のようになるはずです。左と右の図は望ましくない結果を示しています。



上記の図から、設計を最適化するためには、A と B の和を最大化し、A と B の差を最小化したいことがわかります。

この場合のオペランドリストは以下のようになります。

ID	Type	Opt.	Surface	Range	Weight	Target value
01	Flux	Sum	DetectorA		0.0	0.0
02	Flux	Sum	DetectorB		0.0	0.0
03	User-defined	Defined			1.0	(01+02)-1.0
04	User-defined	Defined			1.0	abs(01-02)

Mathmatic functions 数学関数

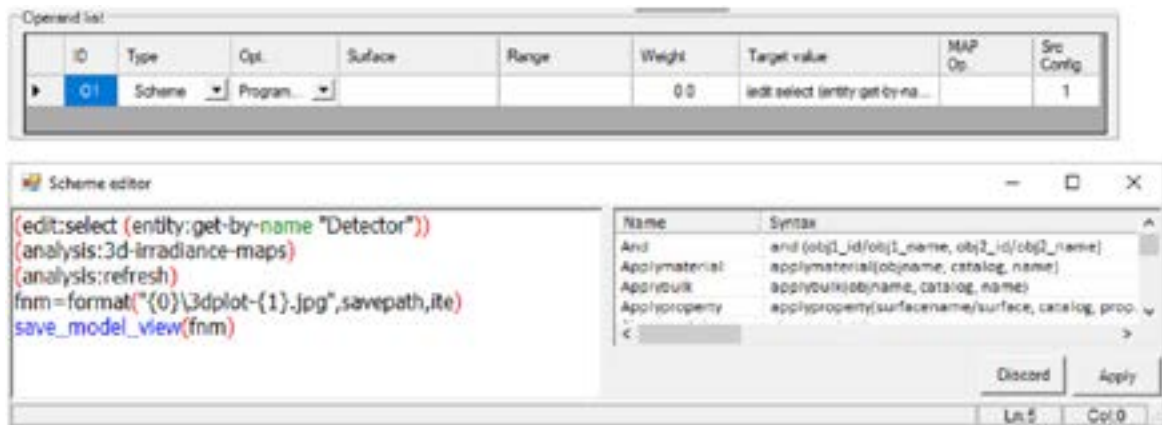
以下は、ピックアップ、ユーザー定義オペランド、アフタースキーム、および「Uniformity 均一度」オペランドで利用できるサポートされている数学関数です。

Sin(rad):	指定された角度（ラジアン単位）の正弦
Cos(rad):	指定された角度（ラジアン単位）の余弦
Tan(rad):	指定された角度（ラジアン単位）の正接
Log(num):	指定された数の底が 10 の対数
Ln(num):	指定された数の自然対数
Exp(num):	e の指定されたべき乗
Sqrt(num):	指定された数の平方根
Abs(num):	指定された数の絶対値
Max(v1,v2):	2 つの数のうち大きい方の値を取得
Min(v1,v2):	2 つの数のうち小さい方の値を取得
Pi():	円周率 (π)
Len(str):	指定された文字列の長さを取得
Len(vec):	指定されたベクトルの長さを取得
Vector(x,y,z):	ベクトルを定義
Point(x,y):	点を定義
Position(x,y,z):	位置を定義
Cstr(value):	指定された数値を文字列に変換

Scheme スキーム

Scheme オペランドは、ユーザーがアフタースキームまたはスキームコマンドを入力するためのスペースです。これらのコマンドは、光線追跡が完了するたびに実行されます。

この例は、各イテレーションの光線追跡終了直後に TracePro のモデルビューを保存する方法を示すものです。

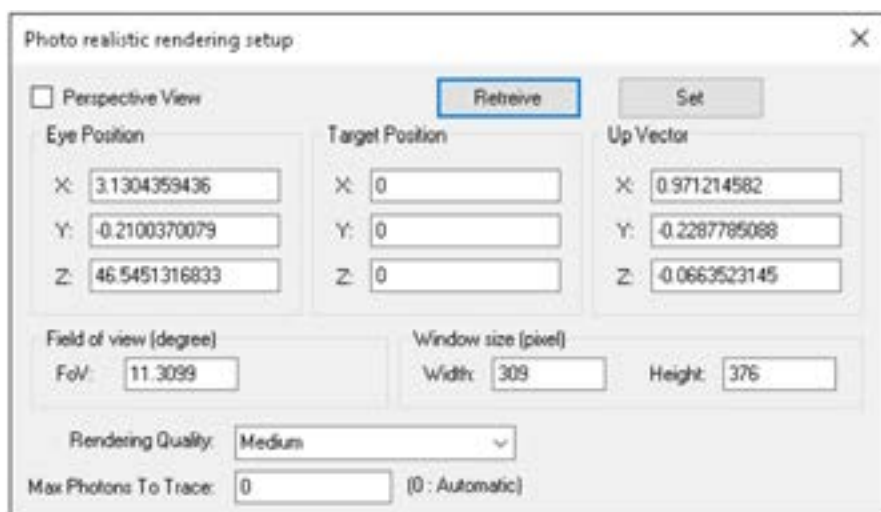


Photorealistic フォトリアリスティック

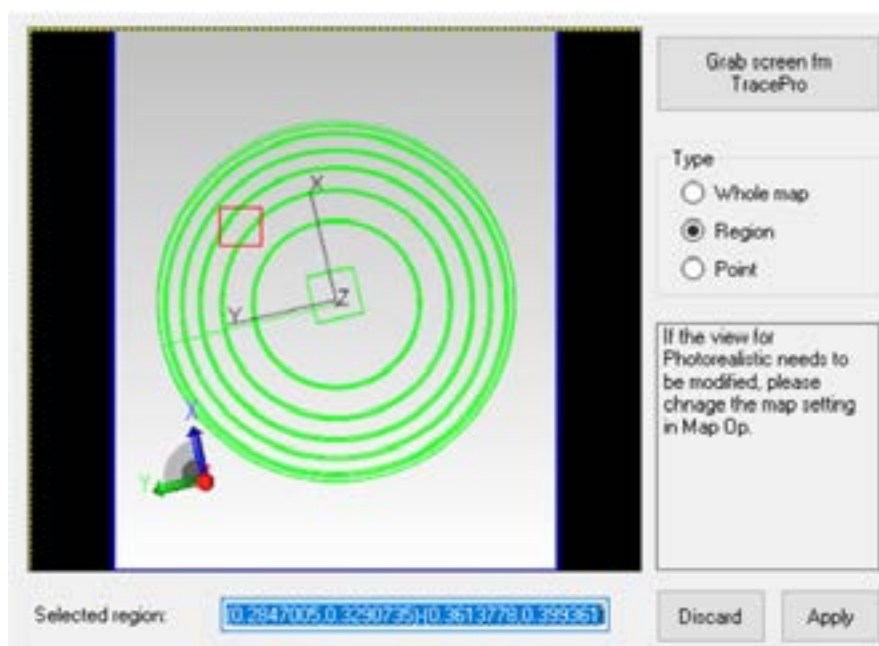
TracePro におけるフォトリアリスティックレンダリングは、散乱挙動を示す光学系の輝度解析結果を効率的に得るための方法を提供します。フォトリアリスティックオペランドは、シミュレーションされたフォトリアリスティックレンダリング結果から輝度値を取得するためのものです。

TracePro でフォトリアリスティックレンダリングを実行するための操作手順を満たすには、範囲とマップ操作でデータを入力するために、フォトリアリスティックオペランドの完全な設定が必要です。推奨される順序は、最初にマップ操作を設定し、次いで範囲セルを設定することです。

マップ操作は、モデルの表示設定を行うためのものであり、フォトリアリスティックレンダリング結果の生成にも使用されます。ダイアログではモデルの表示だけでなく、視野角 (FoV) とウィンドウサイズもここで設定できます。レンダリング品質と追跡する最大フォトン数は、フォトリアリスティックレンダリング設定ダイアログにおける結果の品質を制御するためのパラメータです。



「Range 領域」ダイアログでは、まず TracePro から現在のビューポート設定を取得する必要があります。これは、「Grab screen fm TracePro」ボタンを押すことで実行できます。



照度、カンデラ、輝度のオペランドにおける同様の操作として、3種類のターゲット領域が選択可能です：Whole map 全マップ、Region 領域、Point 点です。最適化の種類 (Opt. の選択) に応じて、ターゲットは平均、最大、最小、RMS の中から選択できます。

After-Scheme（アフタースキーム）プログラミング

Scheme 言語は、TracePro でより複雑なジョブを実現するために提供されている強力なツールです。

After-Scheme は、インタラクティブオプティマイザにおいて同じ目的で使用されます。TracePro に組み込まれている Scheme コマンドに加えて、インタラクティブオプティマイザは、さまざまな目的のための複数の関数もサポートしています。

After-Scheme のエディタは、最適化ダイアログのオブジェクトリストの最後の列をクリックすることで開きます。Scheme コマンドをエディタに入力できるだけでなく、After-Scheme コマンドもここで利用できます。

3D オプティマイザ関数

cplx

アクション：

特定の制御点のローカルな X 値を取得

構文：

```
cplx(cp_id,seg_id,surf_id)
```

引数：

cp_id 整数 (指定された制御点の ID)

seg_id 整数 (制御点が属するセグメントの ID)

surf_id 整数 (セグメントが属するサーフェスの ID)

戻り値：

倍精度浮動小数点数 (double)

説明：

指定された制御点のローカルな X 値を取得します。

cply

アクション：

特定のコントロールポイントのローカル y 値を取得

構文：

```
cply(cp_id,seg_id,surf_id)
```

引数：

cp_id 整数 (指定されたコントロールポイントの ID)

seg_id 整数 (コントロールポイントが属するセグメントの ID)

surf_id 整数 (セグメントが属するサーフェスの ID)

戻り値：

倍精度浮動小数点数 (double)

説明：

指定されたコントロールポイントのローカル y 値を取得します。

cplz

機能：

特定のコントロールポイントのローカル Z 値を取得します。

構文：

```
cplz(cp_id,seg_id,surf_id)
```

引数：

cp_id: 整数 (指定されたコントロールポイントの ID)

seg_id: 整数 (コントロールポイントが属するセグメントの ID)

surf_id: 整数 (セグメントが属するサーフェスの ID)

戻り値:

倍精度浮動小数点数

説明:

指定されたコントロールポイントのローカル Z 値を取得します。

cpx

機能:

特定の制御点の X 座標の値を取得します。

構文:

cpx(cp_id, seg_id, surf_id)

引数:

cp_id: 整数型 (指定された制御点の ID)

seg_id: 整数型 (制御点が属するセグメントの ID)

surf_id: 整数型 (セグメントが属するサーフェスの ID)

戻り値:

倍精度浮動小数点数型

説明:

指定された制御点の X 座標の値を取得します。

cpy

機能:

指定された制御点の Y 座標値を取得します。

構文:

cpy(cp_id, seg_id, surf_id)

引数:

cp_id: 整数 (指定された制御点の ID)

seg_id: 整数 (制御点が属するセグメントの ID)

surf_id: 整数 (セグメントが属するサーフェスの ID)

戻り値:

倍精度浮動小数点数

説明:

指定された制御点の Y 座標値を取得します。

cpz

アクション:

特定の制御点の Z 値を取得

構文:

```
cpz(cp_id, seg_id, surf_id)
```

引数:

cp_id: 整数 (指定された制御点の ID)

seg_id: 整数 (制御点が属するセグメントの ID)

surf_id: 整数 (セグメントが属するサーフェスの ID)

戻り値:

倍精度浮動小数点数 (double)

説明:

指定された制御点の Z 値を取得します。

edge

アクション:

エッジを構築する

構文:

```
edge(edge_type,[surf_id,seg_id]|func_id)
```

引数:

edge_type 文字列 ("segment", "seg": セグメント ; "function", "func": 関数)

surf_id 整数 (サーフェス ID)

seg_id 整数 (セグメント ID)

func_id 整数 (関数 ID) 戻り値:

説明: 既存のセグメントまたは関数からエッジを構築します。

face:asymasphere

アクション:

多項式自由曲面を作成する

構文:

```
face:asymasphere(eq|[xorder,yorder,coef])
```

引数:

eq 文字列 (サグ方程式)

xorder 整数のリスト (各項の x の次数)

yorder 整数のリスト (各項の y の次数)

coef 倍精度のリスト (各項の係数)

戻り値:

face (面)

説明：

与えられた多項式によって自由曲面を作成します。face:freeform() と同等です。

例：

```
f=face:asymasphere("0.1*x+0.1*y+0.2*x*y-0.2*x^2+0.3*y^2")
```

は以下と等しい

```
f=face:asymasphere({1,0,1,2,0},{0,1,1,0,2},{0.1,0.1,0.2,-0.2,0.3})
```

face:biconic

アクション：

双円錐面を作成します

構文：

```
face:biconic(cv,cc[,cvy,ccy])
```

引数：

cv 倍精度浮動小数点数 (曲率)

cc 倍精度浮動小数点数 (円錐定数)

cvy 倍精度浮動小数点数 (Y 方向の曲率)

ccy 倍精度浮動小数点数 (Y 方向の円錐定数)

戻り値：

face (面)

説明：

双円錐面を作成します。引数が 2 つだけ与えられた場合、生成される面は円錐面になります。それ以外の場合、面は双円錐面になります。

face:freeform

アクション：

多項式自由曲面を作成する

構文：

```
face:freeform(eq|[xorder,yorder,coef])
```

引数：

eq 文字列 (サグ方程式)

xorder 整数のリスト (各項の x の次数)

yorder 整数のリスト (各項の y の次数)

coef 倍精度のリスト (各項の係数)

戻り値：

face (面)

説明：

与えられた多項式によって自由曲面を作成します。face:asymasphere() と同等です。

例：

f=face:freeform("0.1*x+0.1*y+0.2*x*y-0.2*x^2+0.3*y^2")

は以下と等価です。

f=face:freeform({1,0,1,2,0},{0,1,1,0,2},{0.1,0.1,0.2,-0.2,0.3})

face:planar

アクション: 平面を作成

構文: face:planar()

引数: null

戻り値: face (面)

説明: 平面を作成します。

fpx

アクション:

指定された関数ポイントの X 座標の値を取得します。

構文:

fpx(cp_id, func_id)

引数:

cp_id 整数型 (指定されたコントロールポイントの ID)

func_id 整数型 (コントロールポイントが属する関数の ID)

戻り値:

倍精度浮動小数点数

fpv

機能:

特定の関数点の y 値を取得

構文:

fpv(cp_id, func_id)

引数:

cp_id 整数型 (指定された制御点の ID)

func_id 整数型 (制御点が属する関数の ID)

戻り値:

倍精度浮動小数点数型

説明:

指定された制御点の y 値を取得します。

get_value

アクション:

指定された関数から、与えられた分数位置に対応する値を取得します。

構文：

function:get_value(func_name|func_id, frac_value)

引数：

func_name: 文字列 (関数名)

func_id: 整数 (関数 ID)

frac_value: 倍精度浮動小数点数 (分数位置 0.0 ～ 1.0)

戻り値：

倍精度浮動小数点数

説明：

指定された関数から、与えられた分数位置に対応する値を取得します。

get_controlpoint_position

アクション：

指定されたコントロールポイントまたはセグメントポイントの位置を取得します。

構文：

get_controlpoint_position([ctrl_id,seg_id,surf_id]|[spnt_id,surf_id])

引数：

ctrl_id 整数 (コントロールポイントの ID)

seg_id 整数 (セグメントの ID)

spnt_id 整数 (セグメントポイントの ID。セグメントポイントを取得しようとする場合に使用)

surf_id 整数 (サーフェスの ID)

戻り値：

position (位置)

説明：

指定されたコントロールポイントの位置を取得します。

get_ite

アクション：

現在のイテレーション回数を取得します

構文：

get_ite()

引数：

null

戻り値：

整数

説明：

現在のイテレーション回数を取得します。

get_object_parameter

アクション:

指定されたオブジェクトから割り当てられたプロパティ値を取得します。

構文:

```
get_object_parameter(obj_id, pname)
```

引数:

obj_id 整数 (オブジェクト ID)

pname 文字列 (パラメータ名)

戻り値:

倍精度浮動小数点数 | 文字列 | 整数 | ベクトル

説明:

指定されたオブジェクトから割り当てられたプロパティ値を取得します。

get_savepath

アクション:

保存パスを取得

構文:

```
get_savepath()
```

引数:

null

戻り値:

文字列

説明:

保存パスを取得します。

get_segment_parameter

アクション:

指定されたセグメントから割り当てられたプロパティ値を取得します。

構文:

```
get_segment_parameter(surf_id, seg_id, pname)
```

引数:

surf_id 整数型 (オブジェクト ID)

seg_id 整数型 (セグメント ID)

pname 文字列型 (パラメータ名)

戻り値:

倍精度浮動小数点数型 | 文字列型 | 整数型 | ベクトル型

説明：

指定されたセグメントから割り当てられたプロパティ値を取得します。

get_surface_parameter

アクション：

指定された表面から割り当てられたプロパティ値を取得します。

構文：

```
get_surface_parameter(surf_id, pname)
```

引数：

surf_id 整数 (表面 ID)

pname 文字列 (パラメータ名)

戻り値：

倍精度浮動小数点数 | 文字列 | 整数 | ベクトル

説明：

指定された表面から割り当てられたプロパティ値を取得します。

get_tp_interface

アクション：

オブティマイザーと TracePro 間のインターフェースを取得

構文：

```
get_tp_interface()
```

引数：

null

戻り値：

tp_interface

説明：

オブティマイザーと TracePro 間のインターフェースを取得します。

math:spline_integration_split

処理：

積分値に基づいてスプライン曲線を分割し、値のリストを取得します。

構文：

```
math:spline_integration_split(list_x, list_y, num, fit_num)
```

引数：

list_x 倍精度浮動小数点数のリスト

list_y 倍精度浮動小数点数のリスト

num 分割する数

fit_num スプラインフィッティング点の数

戻り値:

倍精度浮動小数点数のリスト

説明:

スプラインを等間隔に分割することで、値のリストを取得します。

obj_prop

アクション:

指定されたオブジェクトプロパティの値を取得します。

構文:

obj_prop(pname, obj_id)

引数:

pname 文字列 (プロパティの名前)

obj_id 整数 (オブジェクトの ID)

戻り値:

倍精度浮動小数点数 | 整数 | 文字列 | ベクトル

説明:

指定されたオブジェクトプロパティの値を取得します。property:object() と同等です。

object:get_parameter

アクション:

指定されたオブジェクトから割り当てられたプロパティ値を取得します。

構文:

object:get_parameter(obj_id, pname)

引数:

obj_id 整数型 (オブジェクト ID)

pname 文字列型 (パラメータ名)

戻り値:

倍精度浮動小数点数型 | 文字列型 | 整数型 | ベクトル型

説明:

指定されたオブジェクトから割り当てられたプロパティ値を取得します。object:get_parameter() と同等です。

object:set_parameter

アクション:

指定されたオブジェクトの指定されたプロパティに新しい値を設定します。

構文:

object:set_parameter(obj_id, pname, value)

引数:

obj_id 整数 (オブジェクト ID)

pname 文字列 (パラメータ名)

value 倍精度浮動小数点数 | 整数 | 文字列 | ベクトル (パラメータに設定する新しい値)

戻り値:

null

説明:

指定されたオブジェクトの指定されたプロパティに新しい値を設定します。set_object_parameter() と同等です。

path:get_intersect_plane

アクション:

指定されたパスと指定された平面の交点を取得します。

構文:

path:get_intersect_plane([path]|[surf,seg_id],plane_o,plane_n)

引数:

path 文字列 | 整数 (パス名または ID)

surf 文字列 | 整数 (サーフェス名または ID)

seg_id 整数 (指定されたサーフェス内のセグメント ID)

plane_o 位置 (平面の原点)

plane_n ベクトル (平面の法線)

戻り値:

ベクトル

説明:

指定されたパスと指定された平面の交点を取得します。

path:get_length

アクション:

指定されたパスの、指定されたフラクショナル位置における全長を取得します。

構文:

path:get_tangent([path]|[surf,seg_id])

引数:

path string|integer (パスの名前または ID)

surf string|integer (サーフェスの名前または ID)

seg_id integer (指定されたサーフェス内のセグメント ID)

戻り値:

double

説明：

指定されたパスの、指定されたフラクショナル位置における全長を取得します。

path:get_position

アクション：

指定されたパスの、指定された分数位置における座標を取得します。

構文：

path:get_position([path]|[surf,seg_id],frac_l)

引数：

path string|integer (パスの名前または ID)

surf string|integer (サーフェスの名前または ID)

seg_id integer (指定されたサーフェス内のセグメント ID)

frac_l double (パス内の分数長。0.0 ～ 1.0)

戻り値：

position

説明：

指定されたパスの、指定された分数位置における座標を取得します。

path:get_tangent

アクション：

指定されたパスの、指定されたフラクショナル位置における接線方向を取得します。

構文：

path:get_tangent([path]|[surf,seg_id],frac_l)

引数：

path string|integer (パスの名前または ID)

surf string|integer (サーフェスの名前または ID)

seg_id integer (指定されたサーフェス内のセグメント ID)

frac_l double (パス内のフラクショナル長。0.0 ～ 1.0)

戻り値：

vector (ベクトル)

説明：

指定されたパスの、指定されたフラクショナル位置における接線方向を取得します。

property:object

アクション：

指定されたオブジェクトプロパティの値を取得します

構文：

property:object(pname,obj_id)

引数：

pname 文字列 (プロパティの名前)

obj_id 整数 (オブジェクトの ID)

戻り値：

倍精度浮動小数点数 | 整数 | 文字列 | ベクトル

説明：

指定されたオブジェクトプロパティの値を取得します。obj_prop() と同等です。

property:segment

アクション：

指定されたセグメントプロパティの値を取得します。

構文：

property:segment(pname,seg_id,surf_id)

引数：

pname 文字列 (プロパティの名前)

seg_id 整数 (セグメントの ID)

surf_id 整数 (サーフェスの ID)

戻り値：

実数 | 整数 | 文字列 | ベクトル

説明：

指定されたセグメントプロパティの値を取得します。seg_prop() と同等です。

property:surface

アクション：

指定された表面プロパティの値を取得します。

構文：

property:surface(pname,surf_id)

引数：

pname 文字列 (プロパティの名前)

surf_id 整数 (表面の ID)

戻り値：

double|integer|string|vector (倍精度浮動小数点数 | 整数 | 文字列 | ベクトル)

説明：

指定された表面プロパティの値を取得します。surf_prop() と同等です。

seg_prop

アクション:

指定されたセグメントプロパティの値を取得します。

構文:

```
seg_prop(pname, seg_id, surf_id)
```

引数:

pname : 文字列 (プロパティの名前)

seg_id : 整数 (セグメントの ID)

surf_id : 整数 (サーフェスの ID)

戻り値:

倍精度浮動小数点数 | 整数 | 文字列 | ベクトル

説明:

指定されたセグメントプロパティの値を取得します。property:segment() と同等です。

segment:set_parameter

アクション:

指定されたセグメントの指定されたプロパティに新しい値を設定します。

構文:

```
segment:set_parameter(surf_id, seg_id, pname, value)
```

引数:

surf_id 整数 (オブジェクト ID)

seg_id 整数 (セグメント ID)

pname 文字列 (パラメータ名)

value 倍精度浮動小数点数 | 整数 | 文字列 | ベクトル (パラメータに設定する新しい値)

戻り値:

null

説明:

指定されたセグメントの指定されたプロパティに新しい値を設定します。set_segment_parameter() と同等です。

set_controlpoint_position

アクション:

指定されたコントロールポイントまたはセグメントポイントの位置を設定します。

構文:

```
set_controlpoint_position([ctrl_id,seg_id,surf_id,pos]|[spnt_id,surf_id,pos])
```

引数:

ctrl_id 整数型 (コントロールポイントの ID)

seg_id 整数型 (セグメントの ID)

spnt_id 整数型 (セグメントポイントの ID、セグメントポイントを取得しようとする際に使用)

surf_id 整数型 (サーフェスの ID)

pos ベクトル型 | 位置型 (新しい位置)

戻り値:

null

説明:

指定されたコントロールポイントの位置を取得します。

set_object_parameter

アクション:

指定されたオブジェクトの指定されたプロパティに新しい値を設定します。

構文:

set_object_parameter(obj_id, pname, value)

引数:

obj_id 整数 (オブジェクト ID)

pname 文字列 (パラメータ名)

value 倍精度浮動小数点数 | 整数 | 文字列 | ベクトル (パラメータに設定する新しい値)

戻り値:

null

説明:

指定されたオブジェクトの指定されたプロパティに新しい値を設定します。

set_segment_parameter

アクション:

指定されたセグメントの指定されたプロパティに新しい値を設定します。

構文:

set_segment_parameter(surf_id, seg_id, pname, value)

引数:

surf_id 整数型 (オブジェクト ID)

seg_id 整数型 (セグメント ID)

pname 文字列型 (パラメータ名)

value 倍精度浮動小数点数型 | 整数型 | 文字列型 | ベクトル型 (パラメータに設定する新しい値)

戻り値:

null (何も返しません)

説明:

指定されたセグメントの指定されたプロパティに新しい値を設定します。segment:set_parameter() と同等です。

set_surface_parameter

アクション:

指定されたサーフェスの指定されたプロパティに新しい値を設定します。

構文:

set_surface_parameter(surf_id, pname, value)

引数:

surf_id 整数 (サーフェス ID)

pname 文字列 (パラメータ名)

value 倍精度浮動小数点数 | 整数 | 文字列 | ベクトル (パラメータに設定する新しい値)

戻り値:

null

説明:

指定されたサーフェスの指定されたプロパティに新しい値を設定します。surface:set_parameter() と同等です。

splx

アクション:

特定のセグメントポイントのローカルな X 座標の値を取得します。

構文:

splx(sp_id, surf_id)

引数:

sp_id: 整数 (指定されたセグメントコントロールの ID)

surf_id: 整数 (セグメントポイントが属するサーフェスの ID)

戻り値:

倍精度浮動小数点数

説明:

指定されたセグメントポイントのローカルな X 座標の値を取得します。

sply

アクション:

特定のセグメント点のローカルな Y 値を取得

構文:

sply(sp_id, surf_id)

引数:

sp_id 整数 (指定されたセグメント制御点の ID)

surf_id 整数 (セグメント点が属するサーフェスの ID)

戻り値:

倍精度浮動小数点数 (double)

説明:

特定のセグメント点のローカルな Y 値を取得します。

splz

アクション:

特定のセグメント点のローカル Z 値を取得

構文:

splz(sp_id,surf_id)

引数:

sp_id: 整数 (指定されたセグメント制御点の ID)

surf_id: 整数 (セグメント点が属するサーフェスの ID)

戻り値:

倍精度浮動小数点数

説明:

特定のセグメント点のローカル Z 値を取得します。

spx

アクション:

特定のセグメントポイントの x 値を取得

構文:

spx(sp_id,surf_id)

引数:

sp_id 整数 (指定されたセグメント制御の ID)

surf_id 整数 (セグメントポイントが属するサーフェスの ID)

戻り値:

倍精度浮動小数点数

説明:

特定のセグメントポイントの x 値を取得します。

spy

アクション:

特定のセグメント点の y 値を取得します。

構文:

spy(sp_id,surf_id)

引数：

sp_id 整数 (指定されたセグメント制御点の ID)

surf_id 整数 (セグメント点が属するサーフェスの ID)

戻り値：

double (倍精度浮動小数点数)

説明：

特定のセグメント点の y 値を取得します。

spz

アクション：

特定のセグメントポイントの Z 値を取得

構文：

spz(sp_id,surf_id)

引数：

sp_id: 整数 (指定されたセグメントコントロールの ID)

surf_id: 整数 (セグメントポイントが属するサーフェスの ID)

戻り値：

倍精度浮動小数点数 (double)

説明：

特定のセグメントポイントの Z 値を取得します。

surf_prop

アクション：

指定された表面プロパティの値を取得します。

構文：

surf_prop(pname,surf_id)

引数：

pname 文字列 (プロパティの名前)

surf_id 整数 (表面の ID)

戻り値：

倍精度浮動小数点数 | 整数 | 文字列 | ベクトル

説明：

指定された表面プロパティの値を取得します。property:surface() と同等です。

surface:set_parameter

アクション：

指定されたサーフェスの指定されたプロパティに新しい値を設定します。

構文：

surface:set_parameter(surf_id, pname, value)

引数：

surf_id 整数型 (サーフェス ID)

pname 文字列型 (パラメータ名)

value 倍精度浮動小数点数型 | 整数型 | 文字列型 | ベクトル型 (パラメータに設定する新しい値)

戻り値：

null

説明：

指定されたサーフェスの指定されたプロパティに新しい値を設定します。set_surface_parameter() と同等です。

trajectory

アクション：

TracePro の面から軌跡構造を構築します。

構文：

trajectory([tp_face, uv_spline|rays]|[pos_list, norm_list]|[object_nm, rays])

引数：

tp_face: TracePro の面 (face(face_name) から作成)

uv_spline: uv 曲線を表すスプライン構造

rays: 投影された位置と法線のための光線のリスト

pos_list: 制御点の位置としての位置のリスト

norm_list: 制御点の法線としてのベクトルのリスト

object_nm: 光線との交点を計算するオブジェクトの名前

戻り値：

軌跡構造

説明：TracePro の面から軌跡構造を構築します。軌跡は、曲線と位置ベクトルおよび法線ベクトルを組み合わせた構造です。

trajectory:get_coordinate

アクション：

軌跡から座標を取得します。

構文：

trajectory:get_coordinate(traj, frac_length)

引数：

traj 軌跡構造体

frac_length 座標情報を取得する際の軌跡の相対長 (0.0 ~ 1.0)

戻り値：

座標構造体

説明：

軌跡構造体から座標を取得します。座標構造体は、点、上向きベクトル、および法線ベクトルの組み合わせです。

trajectory:get_length

アクション：

指定された軌跡の全長を取得します。

構文：

```
trajectory:get_length(traj)
```

引数：

traj 軌跡構造

戻り値：

double (倍精度浮動小数点数)

説明：

軌跡構造の全長を取得します。

trajectory:get_normal

アクション：

軌跡内の指定された分数長における法線方向を取得します。

構文：

```
trajectory:get_normal(traj, frac_length)
```

引数：

traj 軌跡構造体

frac_length 位置を取得する分数長

戻り値：

ベクトル

説明：

軌跡構造体から法線方向を取得します。

trajectory:get_position

アクション：

軌跡内の指定されたフラクショナル長における位置を取得します。

構文：

```
trajectory:get_position(traj, frac_length)
```

引数：

traj 軌跡構造

frac_length 位置を取得するフラクショナル長

戻り値：

position

説明：

軌跡構造から位置を取得します。

trajectory:get_tangent

アクション：

軌跡内の指定された分数長における接線方向を取得します。

構文：

trajectory:get_tangent(traj, frac_length)

引数：

traj 軌跡構造

frac_length 位置を取得するための分数長

戻り値：

ベクトル

説明：

軌跡構造から接線方向を取得します。

var

アクション：

変数の値を取得

構文：

var(変数名)

引数：

変数名 文字列 (変数名)

戻り値：

倍精度浮動小数点数 (double)

説明：

変数の値を取得します。

その他項目

append

アクション:

リストに新しい要素を追加する

構文:

append(リスト , 値)

引数:

リスト: リストまたは配列

値: オブジェクト

戻り値:

リスト

説明:

入力されたリストの末尾に新しい要素を追加します。戻り値は、変更されたリストそのものです。

apply_bulk

アクション:

ボディにバルク散乱プロパティを適用

構文:

apply_bulk(obj, cat, prop)

引数:

obj 文字列 | 整数 (オブジェクト名または ID)

cat 文字列 (バルク散乱カタログ)

prop 文字列 (バルク散乱プロパティ名)

戻り値:

null

説明:

ボディにバルク散乱プロパティを適用します。applybulk() と同等です。

apply_material

アクション:

ボディに材料特性を適用する

構文:

apply_material(obj,cat,mat)

引数:

obj 文字列 | 整数 (オブジェクト名または ID)

cat 文字列 (材料カタログ)

mat 文字列 (材料名)

戻り値:

null

説明:

ボディに材料特性を適用します。applymaterial() と同等です。

apply_property

アクション:

面に表面プロパティを適用します。

構文:

```
apply_property(face, cat, prop)
```

引数:

face 文字列 | 面 (面の名前、または face() で作成された面オブジェクト)

cat 文字列 (表面プロパティカタログ)

prop 文字列 (表面プロパティ名)

戻り値:

null

説明:

面に表面プロパティを適用します。applyproperty() と同等です。

applybulk

アクション:

ボディにバルク散乱プロパティを適用します

構文:

```
applybulk(obj,cat,prop)
```

引数:

obj 文字列 | 整数 (オブジェクト名または ID)

cat 文字列 (バルク散乱カタログ)

prop 文字列 (バルク散乱プロパティ名)

戻り値:

null

説明:

ボディにバルク散乱プロパティを適用します。apply_bulk() と同等です。

applymaterial

アクション:

ボディに材料特性を適用します。

構文：

`applymaterial(obj,cat,mat)`

引数：

`obj` 文字列 | 整数 (オブジェクト名または ID)

`cat` 文字列 (材料カタログ)

`mat` 文字列 (材料名)

戻り値：

`null`

説明：

ボディに材料特性を適用します。`apply_material()` と同等です。

applyproperty

アクション：

面に表面特性を適用する

構文：

`applyproperty(face,cat,prop)`

引数：

`face` string | `face` (面の名前、または `face()` で作成された面オブジェクト)

`cat` string (表面特性カタログ)

`prop` string (表面特性名)

戻り値：

`null`

説明：

面に表面特性を適用します。`apply_property()` と同等です。

getitem

アクション：

リスト / ベクター / 位置 / 配列から指定されたアイテムを取得します。

構文：

`getitem(リスト | ベクター | 位置, インデックス)`

引数：

リスト リスト | 配列

ベクター ベクター

位置 位置

インデックス 整数

戻り値：

倍精度浮動小数点数 | 文字列 | 整数 | 真偽値

説明：

この関数は、リスト / 配列構造に格納されている値を取得するためのものです。ベクター、位置、および pos2D 変数にも適用できます。

insert

アクション：

指定された位置に新しい要素をリストに挿入します。

構文：

insert(リスト, インデックス, 値)

引数：

リスト リスト | 配列

インデックス 整数

値 実数 | 文字列 | 整数 | 真偽値 | ベクトル | 位置

戻り値：

リスト

説明：

新しい値をリストに挿入します。戻り値は、変更されたリストです。

name

アクション：

オブジェクトに名前を付ける

構文：

name(old_nm|obj_id|obj_nm,face_nm|obj,face_id|face,new_nm)

引数：

old_nm 文字列 (オブジェクトの元の名前)

obj_id 整数 (インタラクティブオブティマイザで定義されたオブジェクト ID)

obj_nm 文字列 (オブジェクト名)

face_nm 文字列 (面名)

obj 文字列 | 整数 (オブジェクト名または ID)

face_id 整数 (面 ID)

face face (face() によって作成された面オブジェクト)

new_nm 文字列 (オブジェクトの新しい名前)

戻り値：

null

説明：

オブジェクトに名前を付けます。

pause

機能：

プログラムを一定時間（ミリ秒単位）一時停止します。

構文：

pause(ms)

引数：

ms 整数型（待機時間、単位はミリ秒）

戻り値：

ブール型

説明：

wait() と同等です。

prop:add_bulk

アクション：

Henyeey-Greenstein 型の体積散乱特性を追加します。

構文：

prop:add_bulk_gegenbauer(cat, prop, tmps, wvs, g, coeff)

引数：

cat 文字列（体積カタログ名）

prop 文字列（体積特性名）

tmps 倍精度浮動小数点数のリスト（各行の温度）

wvs 倍精度浮動小数点数のリスト（各行の波長）

g 倍精度浮動小数点数のリスト（異方性因子 (g))

coeff 倍精度浮動小数点数のリスト（散乱係数 (1/mm))

戻り値：

null

説明：

Henyeey-Greenstein 型の体積散乱特性を追加します。

例：

temp = {300}

wv = {0.54}

g = {0.95}

coeff = {1}

prop:add_bulk_gegenbauer

アクション:

Gegenbauer 型の体積散乱プロパティを追加します。

構文:

```
prop:add_bulk_gegenbauer(cat,prop,tmps,wvs,g,coeff,alpha)
```

引数:

cat 文字列 (体積カタログ名)

prop 文字列 (体積プロパティ名)

tmps 倍精度浮動小数点数のリスト (各ラインの温度)

wvs 倍精度浮動小数点数のリスト (各ラインの波長)

g 倍精度浮動小数点数のリスト (異方性 (g))

alpha 倍精度浮動小数点数のリスト (Alpha)

戻り値:

null

説明:

Gegenbauer 型の体積散乱プロパティを追加します。

prop:add_lambertian_scatter_surface

アクション:

ランバート散乱を示す表面特性を追加します。

構文:

```
prop:add_lambertian_scatter_surface(cat,prop,a,r,t,brdf,btdf)
```

引数:

cat 文字列 (材料カタログ名)

prop 文字列 (材料特性名)

a 倍精度浮動小数点数 (吸収率)

r 倍精度浮動小数点数 (反射率)

t 倍精度浮動小数点数 (透過率)

brdf 倍精度浮動小数点数 (BRDF)

btdf 倍精度浮動小数点数 (BTDF)

戻り値:

null

説明:

ランバート散乱を示す表面特性を追加します。温度は 300K、波長は 0.5461 です。

prop:add_material

アクション:

TracePro データベースに材料プロパティを追加します。

構文:

```
prop:add_material(cat, prop, tmps, wvs, ri, abs)
```

引数:

cat 文字列 (材料カタログ名)

prop 文字列 (材料プロパティ名)

tmps 倍精度浮動小数点数のリスト (各行の温度)

wvs 倍精度浮動小数点数のリスト (各行の波長)

ri 倍精度浮動小数点数のリスト (各行の屈折率)

abs 倍精度浮動小数点数のリスト (各行の吸収率)

戻り値:

null

説明:

TracePro データベースに材料プロパティを追加します。各リストの要素数は同じである必要があり、同じ順序の項目が一組のデータを構成します。

raytraceflag

アクション:

光線追跡フラグのオン / オフを切り替えます

構文:

```
raytraceflag(obj,value)
```

引数:

obj 文字列 | 整数 (オブジェクト名または ID)

value 真偽値 | 整数 | 文字列 (光線追跡のフラグ)

戻り値:

null

説明:

指定されたオブジェクトを光線追跡の対象に追加または削除します。value が整数の場合、1 は真 (オン)、0 は偽 (オフ) を意味します。"#t" および "#f" を真偽値として使用することもできます。

removeat

アクション:

リストから要素を削除する

構文:

```
removeat( リスト , インデックス )
```

引数：

リスト：リストまたは配列

インデックス：整数

戻り値：

リスト

説明：

入力されたリストから、指定されたインデックスの要素を削除します。戻り値は、要素が削除された後のリスト（変更されたリスト）です。

source:grid_beam_orientation

アクション：

グリッド光源のビーム方向を設定

構文：

```
source:grid_beam_orientation(grid_src_name,mode[mode=0])
```

```
source:grid_beam_orientation(grid_src_name,mode[mode=1],x1,y1,z1,x2,y2,z2)
```

```
source:grid_beam_orientation(grid_src_name,mode[mode=2,3,4,5],x1,y1,z1)
```

引数：

grid_src_name 文字列（グリッド光源の名前）

orientation_mode 整数 (0: グリッドに垂直 1: 方向ベクトル 2: オイラー角 (度) 3: オイラー角 (ラジアン) 4: 点に収束 (ファン) 5: 点から発散 (ファン))

x1 倍精度浮動小数点数

y1 倍精度浮動小数点数

z1 倍精度浮動小数点数

x2 倍精度浮動小数点数

y2 倍精度浮動小数点数

z2 倍精度浮動小数点数

戻り値：

真偽値 (boolean)

説明：

モード 0 は、光線が平面光線グリッドに垂直にトレースされる場合に使用します（「グリッドに垂直」オプション）。これは、角度光線分布の基準方向となります。

ビームの方向は、方向ベクトルと上ベクトルを使用するか、オイラー角の回転を指定することによっても設定できます。

さらに、収束または発散する光線の扇状の集まり（ファン）をトレースするために、光線のある点に向ける、またはある点から発散させることができます。

var_exists

アクション:

指定された変数が存在するかどうかを確認します。

構文:

```
var_exists(v)
```

引数:

v 文字列 (確認する変数の名前)

戻り値:

bool (真偽値 - 存在するなら真 (true)、存在しないなら偽 (false))

説明:

指定された変数が存在するかどうかを確認します。

wait

アクション:

プログラムを一定時間 (ミリ秒単位) 一時停止します。

構文:

```
wait(ms)
```

引数:

ms 整数型 (ミリ秒単位の待ち時間)

戻り値:

ブール型 (bool)

説明:

pause() と同等です。

ファイル I/O

close_all_files

アクション:

開いているすべてのファイルリーダーとライターを閉じます。

構文:

close_all_files()

引数:

なし

戻り値:

null

説明:

開いているすべてのファイルリーダーとライターを閉じます。

close_file

アクション:

open_file 関数によって開かれた指定されたファイルアクセサを閉じます。

構文:

close_file(fid)

引数:

fid 整数型 (ファイル ID)

戻り値:

ブール型

説明:

open_file 関数によって開かれた指定されたファイルアクセサを閉じます。

delete_file

アクション:

指定された場所からファイルを削除します。

構文:

delete(fnm)

引数:

fnm 文字列

戻り値:

ブール値 (真偽値)

説明:

ファイルを削除します。

end_of_file

アクション:

開いているファイルが終端に達したかどうかを確認します。

構文:

```
end_of_file(fid)
```

引数:

fid int (ファイル識別子)

戻り値:

boolean (真偽値)

説明:

開いているファイルが終端に達したかどうかを確認します。

例:

```
fid = open_file("r", "c:\\temp\\test.txt")
while (!(end_of_file(fid)))
    ln = read_file(fid)
    print ln
end while
close_file(fid)
```

file_exists

アクション:

指定されたファイルが存在するかどうかを確認します

構文:

```
file_exist(fnm)
```

引数:

fnm 文字列

戻り値:

ブール値 (bool)

説明:

指定されたファイルが存在する場合は真 (true) を返し、そうでない場合は偽 (false) を返します。

get_application_folder

アクション:

現在実行中のアプリケーションの実行可能ファイルフォルダを返します。

構文:

```
get_application_folder()
```

引数：

なし

戻り値：

文字列

説明：

現在実行中のアプリケーションの実行可能ファイルフォルダを返します。

get_file_extension

アクション：

入力されたフルパス文字列からファイル拡張子を取得します。

構文：

```
get_file_extension(path)
```

引数：

path 文字列

戻り値：

文字列

説明：

この関数は、入力されたパスからファイル拡張子を抽出します。

get_file_name

アクション：

入力されたフルパス文字列からファイル名を取得します。

構文：

```
get_file_name(path)
```

引数：

path 文字列

戻り値：

文字列

説明：

この関数は、入力されたパスからファイル名のみを抽出します。

get_file_path

アクション：

入力されたフルパス文字列からフォルダを取得します。

構文：

```
get_file_path(path)
```

引数：

path 文字列

戻り値:

文字列

説明:

この関数は、入力されたパスからフォルダを抽出します。

get_files

アクション:

指定されたフォルダ内のすべてのファイル名を取得します。

構文:

get_files(フォルダ , [パターン])

引数:

path 文字列

pattern 文字列

戻り値:

リスト

説明:

フォルダ内のファイル名のリストを返します。

get_up_folder

機能:

入力されたフォルダから、指定された階層数だけ上のフォルダの文字列を返します。

構文:

get_up_folder(フォルダ , 上位階層数)

引数:

フォルダ 文字列 (開始フォルダ)

上位階層数 整数 (現在のフォルダから上の階層数)

戻り値:

文字列

説明:

入力されたフォルダから、指定された階層数だけ上のフォルダの文字列を返します。

make_folder

アクション:

指定されたフォルダが存在しない場合に作成します。

構文:

make_folder(path)

引数：

path (文字列):

作成したいフォルダのパスを指定します。

戻り値：

null (何も返しません)

read_file

アクション：

開かれたファイルから一行を読み込む

構文：

```
read_file(fid)
```

引数：

fid int (ファイル ID)

戻り値：

string

説明：

開かれたファイルから一行を読み込む

例：

```
fid = open_file("r", "c:\temp\test.txt")
while (!(end_of_file(fid)))
ln = read_file(fid)
print ln
end while
close_file(fid)
```

wait_until_file_exists

アクション：

指定されたファイルが存在するまでプログラムを一時停止します。

構文：

```
wait_until_file_exists(fnm)
```

引数：

fnm 文字列

戻り値：

ブール値

説明：

指定されたファイルが存在するまで、実行中のプログラムを中断します。

write_file(fid, ln)

機能：

開いているファイルに一行書き込みます。

構文：

```
write_file(fid, ln)
```

引数：

fid: int (ファイル ID) - 書き込み先の開いているファイルの ID です。open_file 関数によって返されます。
ln: string (書き込む行) - ファイルに書き込む文字列です。行末の改行文字は自動的に追加されません。
必要であれば、文字列に含める必要があります (\n など)。

戻り値：

boolean - 書き込みが成功した場合は true、失敗した場合は false を返します。

説明：

この関数は、指定されたファイル ID (fid) を持つ開いているファイルに、引数 ln で指定された文字列を書き込みます。書き込み操作が成功すると true を、何らかの理由で失敗すると false を返します。

例：

```
fid = open_file("w", "c:\\temp\\test.txt")
for i=0 to 9
    write_file(fid, "Line#" + cstr(i) + "\n") // 各行に改行文字を追加
next
close_file(fid)
```

この例では、まず "c:\\temp\\test.txt" という名前のファイルを書き込みモード ("w") で開き、そのファイル ID を fid 変数に格納しています。次に、for ループを使用して 0 から 9 までの数値を順番に処理し、各数値に対して "Line#" + cstr(i) + "\n" という文字列（例: "Line#0\n"、"Line#1\n" など）を write_file 関数でファイルに書き込んでいます。最後に、close_file 関数でファイルを閉じています。この結果、test.txt ファイルには 10 行のテキストが書き込まれます。

write_text

アクション：

指定されたテキストファイルに、文字列を新しい行として書き込みます。

構文：

```
write_text(fnm, str, Append)
```

引数：

fnm 文字列

str 文字列

Append 真偽値

戻り値：

null

説明：

指定されたテキストファイルに単純に文字列を書き込みます。

ファンダメンタル

{...}

アクション:

リストを作成

構文:

aList = {v0, v1, v2,}

引数:

v: 値

戻り値:

list (リスト)

説明:

リストは変数の集まりです。同じ型のオブジェクトを含む必要はありません。リストの一般的な用途の 1 つは、一連の係数を関数に渡すことです。リストの要素は getitem コマンドで取得できます。リストのインデックスはゼロから始まります。

Example (例):

```
aList = {5, 10, "Hello World", vector(1, 2, 3)}
```

```
print getitem(aList, 1) ;=> 10
```

```
print getitem(aList, 2) ;=> Hello World
```

```
print getitem(getitem(aList, 3), 1) ;=> 2
```

=

アクション:

変数に値を代入する

構文:

var = value

引数:

var: 変数名 (文字列)

value: 代入する値 (任意のデータ型)

戻り値:

null (この操作は値を返しません)

説明:

演算子 = は、変数に値を代入するために使用されます。変数は、値を代入する前に宣言する必要はありません。

break

アクション:

現在の for...next または while...end while ループから脱出する

構文：

break

引数：

なし

戻り値：

なし

説明：

現在のループの実行を中断します。

例 1:

```
for i=0 to 10
  if (i>5) then
    break
  endif
next
```

比較演算子

アクション：

左右の 2 つのオペランド間の関係を比較します

構文：

(A == B), (A > B), (A < B), (A >= B), (A <= B), (A != B)

引数：

A double|integer|string|boolean

B double|integer|string|boolean

戻り値：

boolean

説明：

比較演算子は、2 つのオペランド間の関係を判断するために使用されます。

continue

アクション：

後続のコードを実行せずにループを継続します

構文：

continue

引数：

null

戻り値：

null

説明：

ループを次のイテレーションへ継続します。

例 1:

```
a=""
for i=0 to 10
  if (i==5) then
    continue
  endif
  a=a+cstr(i)+","
next
print a
```

for...next

アクション:

囲まれたプロシージャを繰り返し実行します

構文:

```
for var_i = start_int to end_int [step step_size]
....
next
```

引数:

var_i 変数の名前
start_int 整数 (変数の開始値)
end_int 整数 (変数の終了値)
step_size 整数 (各イテレーションにおける変数の増分値)

戻り値:

null

説明:

for...next は反復構造です。バインドされる変数のセット、開始時の初期化方法、および各イテレーションでの更新方法を指定します。step オプションが省略された場合、デフォルト値は 1 です。

例 1: リストの合計

```
aList = {1, 2, 3, 4}
total = 0
for i = 0 to 3
  total = total + getitem(aList, i)
next
print total
```

if...else...endif

アクション:

実行を分岐します

構文:

```
if ( テスト ) then

  結果
```

```
[else  
  代替 ]  
endif
```

引数：

```
テスト    真偽値  
結果      式  
代替      式
```

戻り値：

```
null
```

説明：

テストが真であれば、結果内の式を実行し、そうでなければ代替内の式を実行します。

例 1:

```
b=10  
if (b<=10) then  
  print "yes"  
else  
  print "no"  
endif
```

論理演算子

アクション：

引数に対してブール演算を実行します

構文：

```
(A && B), (A || B)
```

引数：

```
A      真偽値  
B      真偽値
```

戻り値：

```
真偽値
```

説明：

&& は AND 演算子です。A と B の両方が真の場合に真を返します。A または B のどちらかが偽の場合には偽を返します。

|| は OR 演算子です。A または B のどちらかが真の場合に真を返し、それ以外の場合は偽を返します。

while...end while

アクション：

条件が真の間、while...end while ブロック内のコマンドを実行します

構文：

```
while (テスト)  
  
  式
```

...

...

end while

引数：

テスト 真偽値

戻り値：

null

説明：

テストが真の場合、while...end while ブロック内の式を実行します。

例 1:

a=0

while (a<10)

 a=a+1

end while

print a

数学演算子

abs

アクション: 入力データの絶対値を返します

構文:

`abs(value)`

引数:

`value` `double|integer|vector|position`

戻り値:

`double|integer`

説明:

入力データが数値の場合、戻り値はその絶対値です。入力データがベクトルまたは位置の場合、戻り値はその長さです。

acos

アクション:

ラジアンでアークコサイン角を計算します

構文:

`acos(value)`

引数:

`value` `double`

戻り値:

`double`

説明:

アークコサイン値を計算します

acosd

アクション:

アークコサインの角度を度で計算します

構文:

`acosd(value)`

引数:

`value` `double`

戻り値:

`double`

説明:

アークコサインの値を計算します

asin

アクション:

ラジアンでアークサインの角度を計算します

構文:

```
asin(value)
```

引数:

value double

戻り値:

double

説明:

アークサインの値を計算します

asind

アクション:

アークサインの角度を度で計算します

構文:

```
asind(value)
```

引数:

value double (倍精度浮動小数点数)

戻り値:

double (倍精度浮動小数点数)

説明:

アークサインの値を計算します

atan

アクション:

ラジアンでアークタンジェントの角度を計算します

構文:

```
atan(value)
```

引数:

value double

戻り値:

double

説明:

アークタンジェントの値を計算します

atand

アクション:

アークタンジェントの角度を度単位で計算します

構文:

```
atand(value)
```

引数:

value double

戻り値:

double

説明:

アークタンジェントの値を計算します

average

アクション:

入力値の平均値を返します

構文:

```
average( リスト |a,b,c...)
```

引数:

リスト double|integer|vector|position のリスト

a,b,c double|integer|vector|position

戻り値:

double|integer|vector|position

説明:

入力引数がリストの場合、戻り値はリストのすべての要素の平均値です。

入力が一連の値の場合、それらの値の平均値を返します。avg() と同等です。

avg

アクション:

入力値の平均を返します

構文:

```
avg( リスト |a,b,c...)
```

引数:

リスト double|integer|vector|position のリスト

a,b,c double|integer|vector|position

戻り値:

double|integer|vector|position

説明:

入力引数がリストの場合、戻り値はリストのすべての要素の平均です。
入力が連続した値の場合、それらの値の平均を返します。average() と同等です。

cint

アクション:

入力値を整数に変換します

構文:

cint(value)

引数:

value: double (浮動小数点数) または string (文字列)

戻り値:

整数 (integer)

説明:

入力値を整数に変換します

coordinate

アクション:

与えられた原点の位置、上方向ベクトル、法線ベクトルから座標構造を作成します

構文:

coordinate(o, up, norm)

引数:

o 位置 (座標系の原点の位置)

up ベクトル (座標系の上方向ベクトル)

norm ベクトル (座標系の法線ベクトル)

戻り値:

座標

説明:

与えられた原点の位置、上方向ベクトル、法線ベクトルから座標構造を作成します。

coordinate:normal

アクション:

座標系の法線ベクトルを取得します

構文:

coordinate:normal(coord)

引数:

coord 座標

戻り値:

ベクトル

説明：

座標系の法線ベクトルを取得します。

coordinate:origin

アクション：

座標の原点の位置を取得します

構文：

coordinate:origin(coord)

引数：

coord 座標

戻り値：

位置

説明：

座標の原点の位置を取得します。

coordinate:up

アクション：

座標系のアップベクトルを取得します

構文：

coordinate:up(coord)

引数：

coord 座標

戻り値：

ベクトル

説明：

座標系のアップベクトルを取得します。

COS

アクション：

入力された角度（ラジアン）のコサイン値を計算します

構文：

cos(rad)

引数：

rad double（倍精度浮動小数点数）

戻り値：

double（倍精度浮動小数点数）

説明：

コサイン値を計算します

cosd

アクション：

入力角度（度）のコサイン値を計算します

構文：

cosd(deg)

引数：

deg double

戻り値：

double

説明：

コサイン値を計算します

cross

アクション：

2つのベクトルの外積を返します

構文：

cross(va, vb)

引数：

va ベクトル

vb ベクトル

戻り値：

ベクトル

説明：

2つのベクトルの外積を返します。

cstr

アクション：

数値型の値を文字列型に変換します

構文：

cstr(value)

引数：

value double|integer|vector|position

戻り値：

文字列

説明：

数値型の値を文字列型に変換します

dot

アクション：

2つのベクトル / 位置 / 2D 位置のドット積を返します

構文：

dot(va, vb)

引数：

va ベクトル / 位置 / 2D 位置

vb ベクトル / 位置 / 2D 位置

戻り値：

double

説明：

2つのベクトル / 位置 / 2D 位置のドット積を返します。

exp

アクション：

指定されたべき乗に対する e の値を返します

構文：

exp(power)

引数：

power double

戻り値：

double

説明：

指定されたべき乗に対する e の値を返します

get_length

アクション：

ベクトル / 位置 / 2D 位置点の入力リストの全長を取得します

構文：

get_length(list)

引数：

list ベクトル | 位置 | pos2D のリスト

戻り値：

double

説明：

ベクトル / 位置 / 2D 位置点の入力リストの全長を取得します

get_position

アクション：

入力されたベクトル / 位置 / 2D 位置のリストにおいて、指定された割合の長さの位置を取得します

構文：

```
get_position(list, l)
```

引数：

list ベクトル | 位置 | pos2D のリスト

l 割合の長さ (0.0 ~ 1.0)

戻り値：

ベクトル | 位置 | pos2d

説明：

入力されたベクトル / 位置 / 2D 位置のリストにおいて、指定された割合の長さの位置を取得します

get_projected_vector

アクション：

特定の法線ベクトルで定義された平面への投影ベクトルを取得します

構文：

```
get_projected_vector(v, plane_norm)
```

引数：

v ベクトル

plane_norm ベクトル

戻り値：

ベクトル

説明：

特定の法線ベクトルで定義された平面への投影ベクトルを取得します。

get_tangent

アクション：

入力されたベクトル / 位置 / 2D 位置のリストの、指定されたフラクショナル長における方向を取得します

構文：

```
get_tangent(list, l)
```

引数：

list ベクトル | 位置 | pos2D のリスト

l フラクショナル長 (0.0 ~ 1.0)

戻り値：

ベクトル | 位置 | pos2d

説明：

入力されたベクトル / 位置 / 2D 位置のリストの、指定されたフラクショナル長における方向を取得します

iseven

アクション：

テスト値が偶数の整数の場合に真を返し、奇数の場合に偽を返します

構文：

iseven(value)

引数：

value 整数 (テスト値)

戻り値：

真偽値

説明：

テスト値が偶数の整数の場合に真を返し、奇数の場合に偽を返します。

isodd

アクション：

テスト値が奇数の整数の場合に真を返し、偶数の場合に偽を返します

構文：

isodd(value)

引数：

value 整数 (テスト値)

戻り値：

真偽値

説明：

テスト値が奇数の整数の場合に真を返し、偶数の場合に偽を返します。

ln

アクション：

ネイピア数を底とする対数値を計算します

構文：

ln(value)

引数：

value 実数

戻り値:

実数

説明:

ネイピア数を底とする対数値を計算します

log

アクション:

10 を底とする対数値を計算します

構文:

log(value)

引数:

value 実数型 (double)

戻り値:

実数型 (double)

説明:

10 を底とする対数値を計算します。

math:rotate

アクション:

回転操作後の、元のベクトルから得られる結果ベクトルを計算します

構文:

math.rotate(vec, rot_axis, rot_ang)

引数:

vec ベクトル (元のベクトル)

rot_axis ベクトル (回転軸)

rot_ang double (回転角度、単位は度)

戻り値:

ベクトル

説明:

回転計算は、回転中心が原点 (0,0,0) にあることを前提としています。

math:rotate_vector

アクション:

元の向きから目的の向きへの変換に必要な回転で物体を回転させます

構文:

math.rotate_vector (vector, rot_axis, ang)

引数:

vector ベクトル (元のベクトル)

rot_axis ベクトル (回転軸ベクトル)
ang double (回転角度、単位は度)

戻り値：

null

説明：

指定されたベクトルを回転軸の周りに回転させます。

math:spline_integration_split

アクション：

スプライン曲線を積分値に基づいて分割し、値のリストを取得します。

構文：

math:spline_integration_split(list_x, list_y, num, fit_num)

引数：

list_x 倍精度浮動小数点数のリスト
list_y 倍精度浮動小数点数のリスト
num 分割数
fit_num スプラインフィット点の数

戻り値：

倍精度浮動小数点数のリスト

説明：

スプラインを等間隔に分割することで、値のリストを取得します。

max

アクション：

最大値を返します

構文：

max(list|a,b)

引数：

list リスト
a double|integer|vector|position
b double|integer|vector|position

戻り値：

入力と同じ型

説明：

最大値を返します。入力データがベクトルまたは位置の場合、より長い長さを持つものを返します。

min

アクション：

最小値を返します

構文：

`min(list|a,b)`

引数：

`list` リスト

`a` `double|integer|vector|position`

`b` `double|integer|vector|position`

戻り値：

入力と同じ型

説明：

最小値を返します。入力データがベクトルまたは位置の場合、長さが短い方を返します。

normalize

アクション：

正規化されたベクトルを返します

構文：

`normalize(v)`

引数：

`v` ベクトル

戻り値：

ベクトル

説明：

正規化されたベクトルを返します。

pi

アクション：

円周率 (3.1415...) の値を返します

構文：

`pi()`

引数：

なし

戻り値：

浮動小数点数 (double)

説明：

円周率を返します

point

アクション:

点を構築します

構文:

```
point(x,y)
```

引数:

x 整数

y 整数

戻り値:

点

説明:

この関数は点のコンストラクタです

pos

アクション:

3D の位置を構築します

構文:

```
pos(x,y,z|theta,phi|vec)
```

引数:

x double

y double

z double

theta double

phi double

vec vector

戻り値:

position

説明:

この関数は 3D の位置のコンストラクタです。x, y, z を入力するか、または theta (度) と phi (度) を入力できます。position と同等です。

pos2d

アクション:

2D の位置を構築します

構文:

```
pos(x,y,z|theta,phi)
```

引数:

x 実数

y 実数

戻り値:

position2d

説明:

この関数は 2D の位置のコンストラクタです。position2d と同等です。

position

アクション:

3D の位置を構築します

構文:

position(x,y,z|theta,phi|vec)

引数:

x double

y double

z double

theta double

phi double

vec vector

戻り値:

position

説明:

この関数は 3D の位置のコンストラクタです。入力として x, y, z、または theta (度) と phi (度) のいずれかが使用できます。pos と同義です。

position2d

アクション:

2D 位置を構築します

構文:

position2d(x, y, z|theta, phi

引数:

x: 倍精度浮動小数点数 (double) で、2D 位置の X 座標を表します。

y: 倍精度浮動小数点数 (double) で、2D 位置の Y 座標を表します。

戻り値:

position2d

説明:

この関数は 2D 位置のコンストラクタで pos2d と同義です。

ray

アクション:

指定された位置と方向ベクトルから光線構造を作成します

構文:

```
ray(o, dir)
```

引数:

o 位置 (光線の開始位置)

dir ベクトル (光線の方向)

戻り値:

光線構造

説明:

指定された位置と方向ベクトルから光線構造を作成します。

ray:direction

アクション:

光線の方向ベクトルを取得します

構文:

```
ray:direction(ray)
```

引数:

ray 光線

戻り値:

ベクトル

説明:

光線の方向ベクトルを取得します。

ray:is_valid

アクション:

光線方向の長さがゼロの場合に偽 (false) を取得します

構文:

```
ray:is_valid(ray)
```

引数:

ray 光線

戻り値:

真偽値 (bool)

説明:

光線方向の長さがゼロの場合に偽を取得します。

ray:position

アクション:

光線の開始位置を取得します

構文:

```
ray:position(ray)
```

引数:

ray 光線

戻り値:

位置

説明:

光線の開始位置を取得します。

rect

アクション:

整数の矩形変数を作成します

構文:

```
rect(x, y, w, h)
```

引数:

x 整数 (矩形の左端の位置)

y 整数 (矩形の上端の位置)

w 整数 (矩形の幅)

h 整数 (矩形の高さ)

戻り値:

矩形

説明:

整数の矩形変数を作成します。

reset_random

アクション:

乱数生成器をリセットします

構文:

```
reset_random(null|seed)
```

引数:

seed 整数

戻り値:

null

説明:

乱数生成器をリセットします。引数が省略された場合、乱数生成器はシステムティックカウントでリセットされます。それ以外の場合は、入力されたシード番号で初期化されます。

rms

アクション:

入力値の二乗平均平方根 (RMS) 値を計算します

構文:

```
rms(list|a,b,c...)
```

引数:

list double|integer|vector|position のリスト

a,b,c double|integer|vector|position

戻り値:

double

説明:

入力引数がリストの場合、戻り値はリストのすべての要素の RMS 値です。

入力が一連の値の場合、入力の RMS 値を返します。

rnd

アクション:

ランダムな数値 (double 型) を取得します

構文:

```
rnd()
```

引数:

null

戻り値:

double

説明:

ランダムな数値を取得します

sd

アクション:

入力の標準偏差値を計算します

構文:

```
sd(list|a,b,c...)
```

引数:

list double|integer|vector|position のリスト

a,b,c double|integer|vector|position

戻り値:

double

説明：

入力引数がリストの場合、戻り値はすべてのリスト要素の標準偏差値です。入力が一連の値の場合、入力の標準偏差値を返します。

sin

アクション：

入力角度（ラジアン）のサイン値を計算します

構文：

sin(rad)

引数：

rad 実数型

戻り値：

実数型

説明：

サイン値を計算します

sind

アクション：

入力角度（度）のサイン値を計算します

構文：

sind(deg)

引数：

deg 倍精度浮動小数点数 (double)

戻り値：

倍精度浮動小数点数 (double)

説明：

サイン値を計算します

spline

アクション：

入力された制御点に基づいてスプラインを生成します

構文：

spline(ctrl_x, ctrl_y, [ctrl_z])

引数：

ctrl_x リスト (制御点の x 座標値)

ctrl_y リスト (制御点の y 座標値)

ctrl_z リスト (制御点の z 座標値) [省略可能]

戻り値：

スプライン

説明：

入力された制御点を持つスプライン構造を生成します。

sqrt

アクション：

指定された値の平方根を返します

構文：

sqrt(value)

引数：

value double

戻り値：

double

説明：

指定された値の平方根を返します

sum

アクション：

すべての入力値を合計します

構文：

sum(リスト |a,b,c...)

引数：

リスト 倍精度浮動小数点数 | 整数 | ベクトル | 位置 のリスト
a,b,c 倍精度浮動小数点数 | 整数 | ベクトル | 位置

戻り値：

倍精度浮動小数点数 | 整数 | ベクトル | 位置

説明：

入力引数がリストの場合、戻り値はリストのすべての要素の合計です。
入力が一連の値の場合、それらの値の合計を返します。

tan

アクション：

入力された角度（ラジアン）のタンジェント値を計算します

構文：

tan(rad)

引数：

rad 倍精度浮動小数点数 (double)

戻り値：

倍精度浮動小数点数 (double)

説明：

タンジェント値を計算します

tand

アクション：

入力された角度（度単位）のタンジェント値を計算します

構文：

tand(deg)

引数：

deg 倍精度浮動小数点数 (double)

戻り値：

倍精度浮動小数点数 (double)

説明：

タンジェント値を計算します

ubound

アクション：

入力リストの最後のインデックスを返します

構文：

ubound(value)

引数：

value リスト

戻り値：

整数

説明：

配列のインデックスは 0 から始まるため、戻り値はリストの長さから 1 を引いた値になります。

vec

アクション：

3D ベクトルを構築します

構文：

vec(x,y,z|theta,phi|pos)

引数：

x 実数

y 実数

z 実数

theta 実数

phi 実数

pos 位置

戻り値：

ベクトル

説明：

この関数はベクトルのコンストラクタです。x, y, z を入力するか、またはシータ (度) とファイ (度) を入力できます。vector と同等です。

vector

アクション：

3D ベクトルを生成します

構文：

vector(x,y,z|theta,phi|pos)

引数：

x double

y double

z double

theta double

phi double

pos position

戻り値：

vector

説明：

この関数はベクトルのコンストラクタです。直交座標系の x, y, z を入力するか、または極座標系の theta (度) と phi (度) を入力できます。position 型の入力も可能です。vec と同等です。

文字列関数

asc

アクション:

入力文字列の ASCII コードを返します

構文:

asc(ch)

引数:

ch 文字列

戻り値:

整数

説明:

入力文字の ASCII コードを返します。入力文字列が単一文字よりも長い場合、戻り値はその文字列の最初の文字のコードです。

chr

アクション:

入力された ASCII コードに対応する文字を返します

構文:

chr(code)

引数:

code 整数

戻り値:

文字列

説明:

入力された ASCII コードに対応する文字を返します。

instr

アクション:

ターゲット文字列内で検索文字列が見つかった位置を返します

構文:

instr(idx, str, search_str)

引数:

idx 整数 (検索を開始する開始位置)

str 文字列 (ターゲット文字列)

search_str 文字列 (検索する文字列)

戻り値:

整数

説明：

返される位置は 0 から始まります。

lcase

アクション：

入力文字列を小文字に変換します

構文：

lcase(str)

引数：

str 文字列

戻り値：

文字列

説明：

入力文字列の小文字を返します。

len

アクション：

入力引数の長さを返します

構文：

len(str|list|vec|pos)

引数：

str 文字列

list リスト

vec ベクトル

pos 位置

戻り値：

整数 | 倍精度浮動小数点数

説明：

渡された引数が文字列またはリストの場合、戻り値は文字列の長さまたはリストの要素数です。引数がベクトルまたは位置(2 次元または 3 次元)の場合、戻り値はベクトルの長さを表す倍精度浮動小数点数です。

mid

アクション：

元の文字列から、指定された位置と長さの部分文字列を返します

構文：

mid(str, start, length)

引数：

str 文字列 (元の文字列)

start 整数 (部分文字列の開始位置)

length 整数 (部分文字列の長さ)

戻り値：

文字列

説明：

部分文字列の位置は 0 から始まります。

replace

アクション：

指定されたパターンを置換した修正済み文字列を返します

構文：

replace(origin, old_str, new_str)

引数：

origin 文字列 (元の文字列)

old_str 文字列 (置換されるパターン)

new_str 文字列 (置換するパターン)

戻り値：

文字列

説明：

指定されたパターンを置換した修正済み文字列を返します。

ucase

アクション：

入力文字列を大文字に変換します

構文：

ucase(str)

引数：

str 文字列

戻り値：

文字列

説明：

入力された文字列の大文字版を返します。

TracePro の解析にかかわる関数

get_scheme_value()

アクション:

Scheme コマンドの実行結果として返す値を文字列として取得します

構文:

get_scheme_value(scm)

引数:

scm: "string (scheme command that is executed in TracePro)"

戻り値:

string

説明:

1. Scheme コマンドの実行: 引数として渡された Scheme コマンド (scm) を TracePro の Scheme インタープリタに送信して実行します。
2. 戻り値の取得: TracePro が Scheme コマンドの実行結果として返す値を取得します。
3. 文字列への変換: 取得した戻り値を文字列形式に変換します。これは、Scheme のデータ型（数値、シンボルなど）を汎用的な文字列として返すためです。
4. 戻り値の返却: 変換された文字列を関数の戻り値として返します。

重要な考慮事項:

有効な Scheme コマンド: get_scheme_value() 関数に渡す文字列は、TracePro で有効な Scheme コマンドである必要があります。無効なコマンドを渡すと、エラーが発生するか、予期しない結果が返される可能性があります。

戻り値の型:

関数は常に文字列を返しますが、Scheme コマンドが返す元のデータの型（数値、真偽値、オブジェクトなど）は文字列としてエンコードされています。必要に応じて、返された文字列を適切なデータ型（整数、浮動小数点数など）に変換する必要があります。例に示すように、integer() や double() などの変換関数を使用できます。

TracePro のコンテキスト: この関数は TracePro ソフトウェアのコンテキスト内で使用されることを前提としています。スタンドアロンの Scheme インタープリタや他の環境では動作しない可能性があります。

エラー処理:

関数の説明には明示的なエラー処理に関する記述はありません。実際には、Scheme コマンドの実行中にエラーが発生した場合に、適切なエラー処理（エラーを示す文字列を返す、例外を発生させるなど）が行われる可能性があります。

get_scheme_value() 関数を使用することで、TracePro の内部情報にアクセスしたり、TracePro の機能をプログラムから制御したりすることが可能になり、高度な自動化やカスタマイズされたワークフローの構築に役立ちます。

// 例:

TracePro のサーフェスの名前を取得する Scheme コマンドを実行し、その結果を取得する
surface_name_command = "(send (find-entity-by-name \"Surface 1\") 'name)"

```
surface_name = get_scheme_value(surface_name_command)
print surface_name
```

// 例:

TracePro のシステム内のエンティティ数を取得する Scheme コマンドを実行し、その結果を取得する

```
entity_count_command = "(send (get-system) 'number-of-entities)"
entity_count_str = get_scheme_value(entity_count_command)
entity_count = integer(entity_count_str) // 必要に応じて文字列を整数に変換
print entity_count
```

// 例:

TracePro の光源の放射束を取得する Scheme コマンドを実行し、その結果を取得する

```
source_flux_command = "(send (find-entity-by-name \"Source 1\") 'total-flux)"
source_flux_str = get_scheme_value(source_flux_command)
source_flux = double(source_flux_str) // 必要に応じて文字列を浮動小数点数に変換
print source_flux
```

save_candela_polar_distribution

アクション:

カンデラ極分布図を保存します

構文:

```
save_candela_polar_distribution(ray_type, face, fnm)
```

引数:

ray_type 文字列 | 整数 ("incident"|"inc"|"i"|1, "exiting"|"exit"|"e"|2, "missed"|"miss"|"m"|0)
face 文字列 | tp_face (カンデラを保存する面)
fnm 文字列 (カンデラマップを保存するファイル名)

戻り値:

null

説明:

カンデラ極分布図を保存します

```
fnm=format("{0}\plot-{1}.bmp", get_savepath(), get_ite())
delete_file(fnm)
save_candela_polar_distribution("incident", "screen", fnm)
wait_until_file_exists(fnm)
```

save_candela_polar_distribution_missed

アクション:

ミスした (オブジェクト・サーフェスと交差しなかった) 光線に対するカンデラ極分布図を保存します

構文:

```
save_candela_polar_distribution_missed(fnm)
```

引数:

fnm 文字列 (カンデラマップを保存するファイル名)

戻り値：

null

説明：

ミスした光線に対するカンデラ極分布図を保存します

```
fnm=format("{0}\plot-{1}.bmp",get_savepath(), get_ite())
delete_file(fnm)
save_candela_polar_distribution_missed(fnm)
wait_until_file_exists(fnm)
```

save_candela_rectangular_distribution

アクション：

カンデラ矩形分布図を保存します

構文：

```
save_candela_rectangular_distribution(ray_type, face, fnm)
```

引数：

ray_type 文字列 | 整数 ("incident"|"inc"|"i"|1, "exiting"|"exit"|"e"|2, "missed"|"miss"|"m"|0)
face 文字列 | tp_face (カンデラを保存する面)
fnm 文字列 (カンデラマップを保存するファイル名)

戻り値：

null

説明：

カンデラ矩形分布図を保存します

```
fnm=format("{0}\plot-{1}.bmp", get_savepath(), get_ite())
delete_file(fnm)
save_candela_rectangular_distribution("incident", "screen", fnm)
wait_until_file_exists(fnm)
```

save_candela_rectangular_distribution_missed

アクション：

ミスした光線に対する矩形極座標配光分布図を保存します

構文：

```
save_candela_rectangular_distribution_missed(fnm)
```

引数：

fnm 文字列 (カンデラマップを保存するファイル名)

戻り値：

null

説明：

ミスした光線に対するカンデラ矩形配光分布図を保存します。

```
fnm=format("{0}\plot-{1}.bmp",get_savepath(), get_ite())
```

```
delete_file(fnm)
save_candela_rectangular_distribution_missed(fnm)
wait_until_file_exists(fnm)
```

save_irradiance_map

アクション:

指定された面における放射照度マップを保存します

構文:

```
save_irradiance_map(face, fnm)
```

引数:

face 文字列 | 面 (TracePro における面名または面オブジェクト)
fnm 文字列 (マップを保存するファイル名)

戻り値:

null

説明:

このコマンドは、選択された面の放射照度マップを開き、保存します。

```
fnm=format("{0}\plot-{1}.bmp", get_savepath(), get_ite())
delete_file(fnm)
save_irradiance_map("screen", fnm)
wait_until_file_exists(fnm)
```

save_iso_candela_polar

アクション:

等カンデラ極座標マップを保存します

構文:

```
save_iso_candela_polar(ray_type, face, fnm)
```

引数:

ray_type 文字列 | 整数 ("incident"|"inc"|"i"|1, "exiting"|"exit"|"e"|2, "missed"|"miss"|"m"|0) (光線の種類)
face 文字列 | tp_face (カンデラを保存する面)
fnm 文字列 (カンデラマップを保存するファイル名)

戻り値:

null

説明:

等カンデラ極座標マップを保存します。

```
fnm=format("{0}\plot-{1}.bmp", get_savepath(), get_ite())
delete_file(fnm)
save_iso_candela_polar("incident", "screen", fnm)
wait_until_file_exists(fnm)
```

save_iso_candela_polar_missed

アクション:

ミスした光線に対する等照度極座標図を保存します

構文:

```
save_iso_candela_polar_missed(fnm)
```

引数:

fnm 文字列 (照度図を保存するファイル名)

戻り値:

null

説明:

ミスした光線に対する等照度極座標図を保存します

```
fnm=format("{0}\plot-{1}.bmp",get_savepath(), get_ite())
delete_file(fnm)
save_iso_candela_polar_missed(fnm)
wait_until_file_exists(fnm)
```

save_iso_candela_rectangular

アクション:

等カンデラ矩形マップを保存します

構文:

```
save_iso_candela_rectangular(ray_type, face, fnm)
```

引数:

ray_type 文字列 | 整数 ("incident"|"inc"|"i"|1, "exiting"|"exit"|"e"|2, "missed"|"miss"|"m"|0)
face 文字列 | tp_face (カンデラを保存する面)
fnm 文字列 (カンデラマップを保存するファイル名)

戻り値:

null

説明:

等カンデラ矩形マップを保存します

```
fnm=format("{0}\plot-{1}.bmp", get_savepath(), get_ite())
delete_file(fnm)
save_iso_candela_rectangular("incident", "screen", fnm)
wait_until_file_exists(fnm)
```

save_iso_candela_rectangular_missed

アクション:

ミスした光線に対する等照度矩形図を保存します

構文:

save_iso_candela_rectangular_missed(fnm)

引数:

fnm 文字列 (カンデラマップを保存するファイル名)

戻り値:

null

説明:

ミスした光線に対する等照度矩形図を保存します

```
fnm=format("{0}\plot-{1}.bmp".get_savepath(), get_ite())
```

```
delete_file(fnm)
```

```
save_iso_candela_rectangular_missed(fnm)
```

```
wait_until_file_exists(fnm)
```

save_model_view

アクション:

モデルビュー画面をビットマップとして保存します

構文:

```
save_model_view(fnm)
```

引数:

fnm 文字列 (モデルビューの保存先のファイル名)

戻り値:

null

説明:

アクティブなモデルビューをファイルとして保存します。注意: ファイル拡張子は .bmp である必要があります。

save_photorealistic_render

アクション:

現在のフォトリアリスティックレンダリングマップをファイルに保存します

構文:

```
save_photorealistic_render(fnm)
```

引数:

fnm 文字列 (フォトリアリスティックマップを保存するファイル名)

戻り値:

null

説明:

現在のフォトリアリスティックレンダリングマップをファイルに保存します。ファイル拡張子によって保存されるファイルの種類が決まります。「.txt」は生データを保存することを意味し、「.bmp」、「.jpg」は結果をビットマップとして保存するためのものです。

set_model_view_window_size

アクション:

TracePro のモデルビューウィンドウを指定されたサイズに設定します

構文:

```
set_model_view_window_size(width, height)
```

引数:

width 整数 (モデルビューウィンドウの希望する幅。単位はピクセル)

height 整数 (モデルビューウィンドウの希望する高さ。単位はピクセル)

戻り値:

null

説明:

TracePro のモデルビューウィンドウを指定されたサイズに設定します。

set_view

アクション:

TracePro のアクティブなモデルビューの視点を設定します

構文:

```
set_view(eye, lookat, up, persp, [width, height])
```

引数:

eye position (視点の位置)

lookat position (注視点 / 画面中央の位置)

up vector (視点の上方向ベクトル)

persp bool|integer|string (透視投影を使用するかどうか)

width double (実際のモデルビューポートの幅)

height double (実際のモデルビューポートの高さ)

戻り値:

null

説明:

TracePro のアクティブなモデルビューの視点を設定します。persp は真偽値、0|1、または "#t"|"#f" を受け付けます。オプションの引数 width と height は、モデルビューポートの実際のサイズを mm 単位で定義するために使用します。

TracePro ファイルに関する関数

close_model()

アクション:

TracePro で現在開いているモデルを閉じます。

構文:

close_model()

引数:

null

戻り値:

null

説明:

TracePro で現在開いているモデルを閉じます。closemodel() と同じ機能です。

closemodel

アクション:

TracePro で現在開いているモデルを閉じます

構文:

closemodel()

引数:

null

戻り値:

null

説明:

TracePro で現在開いているモデルを閉じます。close_model() と同等です。

load_model

アクション:

TracePro に oml モデルをロードします

構文:

load_model(fnm)

引数:

fnm 文字列 (モデルファイル名)

戻り値:

null

説明:

TracePro に oml モデルをロードします。loadmodel() と同等です。

loadmodel

アクション:

TracePro に OML モデルをロードします

構文:

```
loadmodel(fnm)
```

引数:

fnm 文字列 (モデルファイル名)

戻り値:

null

説明:

TracePro に OML モデルをロードします。load_model() と同等です。

TracePro モデリングに関する関数

and

アクション:

2 つのオブジェクトに対して交差のブール演算を実行します

構文:

and(obj1, obj2)

引数:

obj1 文字列 | 整数 (交差演算の最初のオブジェクト)

obj2 文字列 | 整数 (交差演算の 2 番目のオブジェクト)

戻り値:

null

説明:

2 つのオブジェクトに対して交差のブール演算を実行します。オブジェクトを指定するには、整数または文字列のいずれかを使用できます。整数はインタラクティブオブティマイザーで割り当てられたオブジェクト ID を意味し、文字列は TracePro で認識される名前です。intersect() と同等です。

bend

アクション:

TracePro でオブジェクトをカーブに沿って曲げます

構文:

bend(obj, start, end, init_rail, crv)

引数:

obj 文字列 (TracePro 内のターゲットオブジェクト名)

start 位置 (オブジェクトの曲げを開始する位置)

end 位置 (オブジェクトの曲げを終了する位置)

init_rail ベクトル (曲げの初期レールベクトル)

crv 文字列 (TracePro で定義されたエッジの名前)

戻り値:

null

説明:

TracePro でオブジェクトをカーブに沿って曲げます。

body_transform

アクション:

変換行列を用いてボディ (物体) を変換します

構文:

body_transform(obj, mx)

引数:

obj 文字列 | 整数 (変換されるオブジェクトの名前または ID)
mx 変換 (変換行列)

戻り値:

null

説明:

変換行列は transform:* 関数によって作成されます。

copy

アクション:

TracePro 内の既存のオブジェクトをコピーし、新しい名前を付けます

構文:

copy(obj, nm)

引数:

obj 文字列 | 整数 (コピーするオブジェクト)
nm 文字列 (コピーされたオブジェクトの名前)

戻り値:

null

説明:

TracePro 内の既存のオブジェクトをコピーし、新しい名前を付けます

create_edge

アクション:

TracePro に指定されたエッジを作成します

構文:

create_edge([seg_nm]|[surf_id,seg_id],nm)

引数:

seg_nm 文字列 (Interactive Optimizer のセグメント名)
surf_id 整数 (Interactive Optimizer のサーフェス / プロファイルの ID)
seg_id 整数 (Interactive Optimizer のセグメント ID)
nm 文字列 (TracePro での指定名)

戻り値:

null

説明:

TracePro に指定されたエッジを作成します。createedge() と同等です。

create_surface

アクション:

TracePro に指定されたサーフェスを作成します

構文：

```
create_surface(s_id, nm)
```

引数：

s_id 整数 (Interactive Optimizer におけるサーフェス ID)

nm 文字列 (TracePro における指定された名前)

戻り値：

null

説明：

TracePro に指定されたサーフェスを作成します。createsurface() と同等です。

createedge

アクション：

TracePro に指定されたエッジを作成します

構文：

```
createedge([seg_nm]|[surf_id,seg_id],nm)
```

引数：

seg_nm 文字列 (Interactive Optimizer のセグメント名)

surf_id 整数 (Interactive Optimizer のサーフェス / プロファイルの ID)

seg_id 整数 (Interactive Optimizer のセグメントの ID)

nm 文字列 (TracePro での指定名)

戻り値：

null

説明：

TracePro に指定されたエッジを作成します。create_edge() と同等です。

createsurface

アクション：

TracePro に指定されたサーフェスを作成します

構文：

```
createsurface(s_id, nm)
```

引数：

s_id 整数 (インタラクティブオプティマイザーにおけるサーフェス ID)

nm 文字列 (TracePro における指定された名前)

戻り値：

null

説明：

TracePro に指定されたサーフェスを作成します。create_surface() と同等です。

del

アクション:

TracePro のアクティブなモデルから指定されたオブジェクトを削除します

構文:

```
del(obj)
```

引数:

obj 文字列 | 整数 (削除するオブジェクト)

戻り値:

null

説明:

TracePro のアクティブなモデルから指定されたオブジェクトを削除します。delete() と同等です。

delete

アクション:

TracePro のアクティブなモデルから指定されたオブジェクトを削除します

構文:

```
delete(obj)
```

引数:

obj: 削除したいオブジェクトを指定します。

string: オブジェクトの名前 (文字列) で指定できます。

integer: オブジェクトの ID (整数) で指定できます。

戻り値:

null

説明:

TracePro のアクティブなモデルから指定されたオブジェクトを削除します。del() と同等です

export_surface_as_surface

アクション:

指定されたサーフェスを TracePro にサーフェスエンティティとしてエクスポートします

構文:

```
xport_surface_as_surface(surf_id)
```

引数:

surf_id: 整数 (integer) であり、「Interactive Optimizer におけるサーフェス / プロファイル ID」を表します。

エクスポートしたいサーフェスの識別番号を指定します。

戻り値:

null

説明:

指定されたサーフェスを TracePro にサーフェスエンティティとしてエクスポートします。このコマンドは、サーフェスエディタのポップアップメニューを押して TracePro にサーフェスをエクスポートするのと同じです。

この関数は、TracePro のサーフェスエディタの GUI 操作と同じエクスポート処理をプログラムから実行できる機能を提供します。この関数を使うことで、Interactive Optimizer で選択または指定したサーフェスを、その ID を使ってプログラムの TracePro にサーフェスデータとして出力できるということです。これは、スクリプトや自動化されたプロセスの中で TracePro のデータ連携を行う際に役立ちます。

export_surface_as_wirebody

アクション: 指定されたサーフェスをワイヤーボディエンティティとして TracePro にエクスポートします

構文:

```
export_surface_as_wirebody(surf_id)
```

引数:

surf_id: サーフェスまたはプロファイルの ID を表す整数値です。これは、Interactive Optimizer（対話型最適化ツール）内でサーフェスを識別するために使用される番号です。

戻り値:

null エクスポートという操作を実行するだけで、値を返す必要がないため

説明:

指定されたサーフェスをワイヤーボディエンティティとして TracePro にエクスポートします。このコマンドは、サーフェスエディタのポップアップメニューを押して TracePro にサーフェスをエクスポートするのと同じです。TracePro のサーフェスエディタの GUI 操作と同じ機能を提供することを意味しています。ワイヤーボディとは、サーフェスの形状を線（エッジ）だけで表現したエンティティのことです。この関数を使うと、Interactive Optimizer で指定した ID を持つサーフェスを、TracePro にワイヤーボディ形式で自動的にエクスポートできます。これは、手動で GUI 操作を行う代わりに、スクリプトなどからエクスポート処理を自動化する際に便利です。

face

アクション:

TracePro で見つかったサーフェスを参照する仮想オブジェクトを作成します

構文:

```
face([object,face]|entity)
```

引数:

object 文字列 | 整数 (TracePro のオブジェクト名、またはインタラクティブオプティマイザーのオブジェクト ID)

face 文字列 | 整数 (TracePro の面名、または TracePro モデル内の面の番号)

entity 文字列 | 整数 (TracePro 内の一意のエンティティ名または ID)

戻り値:

face

説明:

TracePro で見つかった面を参照する仮想オブジェクトを作成します。

face:centerposition

アクション:

指定された平面ポリゴン面の重心を取得します

構文:

```
face:centerposition(f|[obj,fid]|[o_nm,f_nm])
```

引数:

f 文字列 | サーフェス (サーフェスの名前、または face() で作成された面オブジェクト)

obj 文字列 | 整数 (オブジェクトの名前、またはインタラクティブオブティマイザーで作成されたオブジェクト ID)

fid 整数 (サーフェス ID)

o_nm 文字列 (オブジェクトの名前)

f_nm 文字列 (サーフェスの名前)

戻り値:

位置

説明:

指定された平面ポリゴンサーフェスの重心を取得します。facecenterposition() と同等です。

さまざまな方法で指定された平面ポリゴン面の重心の 3 次元位置情報を取得するためのものです。物理シミュレーション、衝突検出、オブジェクトの操作など、さまざまなアプリケーションで役立ちます。

face:get_normal

アクション:

サーフェスの法線を取得します

構文:

```
face:get_normal(f, pos)
```

引数:

f 文字列 | face (サーフェスの名前、または face() で作成されたサーフェスオブジェクト)

pos position (法線を取得する位置)

戻り値:

vector

説明:

サーフェスの法線を取得します。サーフェスは平坦である必要はなく、曲面でも構いません。

3D モデルの面上の特定の位置における、表面に垂直な方向を示すベクトルを取得するために使用されます。これは、レンダリング、衝突検出、物理シミュレーションなど、さまざまな 3D グラフィックスやシミュレーションの分野で重要な情報となります。

face:load_facet

アクション:

TracePro モデルから光線交差の準備としてファセットをロードします

構文:

```
face:load_facet(f)
```

引数:

f: face 型の引数です。ロードしたい TracePro モデル内の特定の面（フェイス）を指定します。

戻り値:

bool

説明:

TracePro モデルから光線交差の準備としてファセットをロードします。

TracePro のモデルに含まれる特定の「サーフェス」の微細な三角形分割（ファセット）データを読み込むためのものです。これは、その面と光線がどのように交差するかを正確に計算するために必要な準備段階です。戻り値の真偽値は、ロード処理が成功したかどうかを示します。

下の例では、myFace という face 型の変数で表される特定の面に対して、face:load_facet 関数を呼び出しています。これにより、myFace のファセットデータがロードされ、その後の光線追跡計算で使えるようになります。

face:ray_intersect

アクション:

面の交点の位置と法線を計算します。

構文:

```
face:ray_intersect(f, ray)
```

引数:

f: face - 交点を計算する対象の面オブジェクトです。この面は、ロードされたファセット（小さな平面の集合体として表現された 3D 形状の一部）を持っている必要があります。

ray: ray - 面と交差するかどうかをテストする光線オブジェクトです。光線は、原点（開始位置）と方向ベクトルを持ちます。

戻り値:

ray - この関数は、光線構造体を返します。

交点が見つかった場合、返される光線構造体の「原点 (ray.origin())」が交点の 3D 空間における位置を表し、「方向 (ray.direction())」がその交点における面の法線ベクトルを表します。法線ベクトルは、面に対して垂直な方向を示すベクトルです。

交点が見つからなかった場合、返される光線構造体の方向ベクトルはゼロベクトル（例: {0, 0, 0}）になります。これにより、交点が存在しなかったことを呼び出し元のコードで判断できます。

説明:

ロードされたファセットを持つ面と光線の交点の位置と法線を計算します。返される値は光線構造体であり、交点の位置を取得するには ray.origin() を、法線を取得するには ray.direction() を使用します。返された光線の方向ベクトルがゼロベクトルの場合、交点が見つからなかったことを意味します。

```
// 例 1: 面と光線の交点を計算し、交点の位置と法線を出力する
my_face = ... // 何らかの方法で面オブジェクトを取得
my_ray = ... // 何らかの方法で光線オブジェクトを作成

intersection_ray = face:ray_intersect(my_face, my_ray)

if (vector_length(intersection_ray:direction()) > 0) then
    intersect_position = intersection_ray:origin()
    intersect_normal = intersection_ray:direction()
    print " 交点位置: " + str(intersect_position)
    print " 交点法線: " + str(intersect_normal)
else
    print " 光線は面と交差しませんでした。"
endif

// 例 2: 複数の面と 1 つの光線の交点をチェックし、最初の交点の情報を取得する
faces = [...] // 面オブジェクトのリスト
test_ray = ... // 光線オブジェクト

for each face_obj in faces
    intersect_result = face:ray_intersect(face_obj, test_ray)
    if (vector_length(intersect_result:direction()) > 0) then
        first_intersection_position = intersect_result:origin()
        first_intersection_normal = intersect_result:direction()
        print " 最初の交点位置: " + str(first_intersection_position)
        print " 最初の交点法線: " + str(first_intersection_normal)
        break // 最初の交点を見つけたらループを抜ける
    endif
next
```

face:ray_pick

アクション:

指定されたレイによってピックされた面を参照するフェイス仮想オブジェクトを作成します

構文:

face:ray_pick([ray_o, ray_d]|ray) のように、引数は 2 つの形式があります。

- [ray_o, ray_d]: レイの始点 (ray_o) と方向 (ray_d) をそれぞれ position (位置) と vector (ベクトル) で指定する形式。
- ray: 事前に定義された ray 型の変数 ray を直接指定する形式。

引数:

ray_o: レイの始点 (picking ray start position) を示す position 型の値です。レイがどこから始まるかを定義します。

ray_d: レイの方向 (pick ray direction) を示す vector 型の値です。レイがどの方向に進むかを定義します。

ray: ピッキングレイ (picking ray) そのものを示す ray 型の値です。始点と方向の情報が一体となったオブジェクトです。

戻り値：

face

説明：

指定されたレイによってピックされた面を参照するフェイス仮想オブジェクトを作成します。

使用例：

```
picked_face = face:ray_pick({0, 0, 0}, {0, 0, -1}) // 原点から Z 軸負方向に射出したレイがピックした面を参照
```

```
picked_face2 = face:ray_pick(my_picking_ray) // 事前に定義されたレイ変数 'my_picking_ray' がピックした面を参照
```

```
picked_face = face:ray_pick({0, 0, 0}, {0, 0, -1}):
```

レイの始点が原点 {0, 0, 0} で、方向が Z 軸の負方向 {0, 0, -1} であるレイを射出し、そのレイがピックした面を参照するフェイスオブジェクト picked_face を作成します。

```
picked_face2 = face:ray_pick(my_picking_ray):
```

事前に定義された my_picking_ray という ray 型の変数が示すレイがピックした面を参照するフェイスオブジェクト picked_face2 を作成します。

この関数は特定の面を動的に参照し、その属性や形状を操作するために利用されます。

face:uv_bound

アクション：

指定された面のパラメトリックボックスの u_min、u_max、v_min、v_max を持つリストを返します

構文：

```
face:uv_bound(face)
```

引数：

face 文字列 | サーフェス (サーフェスの名前、または face() で作成されたサーフェスオブジェクト)

戻り値：

リスト (uMin, uMax, vMin, vMax)

説明：

指定された面のパラメトリックボックスの u_min、u_max、v_min、v_max を持つリストを返します。

補足：

パラメトリックボックス (Parametric Box): 3D モデルの面は、通常、パラメーター空間 (UV 空間) で定義されます。パラメトリックボックスは、そのパラメーター空間における面の範囲を矩形で表したものです。

UV 座標: 3D モデルの表面上の点を 2 次元の座標 (U 座標と V 座標) で表現する方法です。テクスチャマッピングなどで使用されます。

uMin, uMax, vMin, vMax: それぞれ、U 座標の最小値、U 座標の最大値、V 座標の最小値、V 座標の最大値を意味します。これらの値を知ることで、テクスチャの適用範囲などを制御することができます。

この関数は、3D モデルの特定の面について、その UV 座標の範囲 (最小値と最大値) をリスト形式で取得するためのものです。

facecenterposition

アクション:

指定された平面ポリゴン面の重心を取得します

構文:

```
facecenterposition(f|[obj,fid]|[o_nm,f_nm])
```

引数:

f 文字列 | 面 (面の名前、または face() で作成された面オブジェクト)

obj 文字列 | 整数 (オブジェクトの名前、またはインタラクティブオブティマイザーで作成されたオブジェクト ID)

fid 整数 (面の ID)

o_nm 文字列 (オブジェクトの名前)

f_nm 文字列 (面の名前)

戻り値:

position

説明:

指定された平面ポリゴン面の重心を取得します。face:centerposition() と同等です。

3D モデル内の特定の平面の重心の 3 次元座標を取得するためのものです。オブジェクト名と Surface 名、オブジェクト ID とサーフェス ID、または直接サーフェスオブジェクトを指定することで、目的のサーフェスを特定できます。

grotate

アクション:

グローバル座標でオブジェクトを回転させる

構文:

```
grotate(obj, r_cen, r_axis, r_ang)
```

引数:

obj 文字列 | 整数 (回転させるオブジェクト)

r_cen 位置 (回転中心)

r_axis ベクトル (回転軸)

r_ang 実数 (回転角度、単位は度)

戻り値:

null

説明:

グローバル座標でオブジェクトを回転させます。

指定されたオブジェクトを、グローバル座標系の特定の点を中心として、特定の軸周りに、指定された角度だけ回転させる関数です。

grotate_adv

アクション:

オブジェクトの元の方向から目的の方向への変換に基づいて回転させます

構文:

```
grotate_adv(obj, fromN, fromUp, toN, toUp)
```

引数:

obj 文字列 | 整数 (オブジェクトの名前または ID)

fromN ベクトル (元の法線方向)

fromUp ベクトル (元のアップ方向)

toN ベクトル (目的の法線方向)

toUp ベクトル (目的のアップ方向)

戻り値:

null

説明:

オブジェクトの元の方向から目的の方向への変換に基づいて回転させます

単に角度や軸を指定して回転させるだけでなく、元のオブジェクトの向き（法線ベクトルとアップベクトルで定義）と、回転後の目的の向き（同様に法線ベクトルとアップベクトルで定義）を指定することで、オブジェクトを正確に回転させることができる高度な機能を提供しています。

intersect

アクション:

2つのオブジェクトに対して交差のブール演算を実行します

構文:

```
intersect(obj1, obj2)
```

引数:

obj1 文字列 | 整数 (交差演算の最初のオブジェクト)

obj2 文字列 | 整数 (交差演算の2番目のオブジェクト)

戻り値:

null

説明:

2つのオブジェクトに対して交差のブール演算を実行します。オブジェクトを指定するには、整数または文字列のいずれかを使用できます。整数はインタラクティブオブティマイザで割り当てられたオブジェクト ID を意味し、文字列は TracePro で認識される名前です。and() と同等です。

lrotate

アクション:

オブジェクトを自身のローカル座標で回転させます

構文:

```
lrotate(obj, r_axis, r_ang)
```

引数：

obj: 文字列（オブジェクトの名前）または整数（オブジェクト ID）で指定。

r_axis: 回転軸。3次元ベクトル {x, y, z}。回転はこのベクトルで定義される軸を中心に行われます。

r_ang: 回転角度。度単位の double（浮動小数点数）で指定。正の値は反時計回り（通常は右ねじの法則に従います）、負の値は時計回りの回転を示します。

戻り値：

null

説明：

オブジェクトを自身のローカル座標で回転させます。オブジェクトが元々持っている向きや位置を基準として回転が行われます。

例：

lrotate("Cube1", {0, 1, 0}, 45): 名前が "Cube1" のオブジェクトを、自身のローカル Y 軸（ベクトル {0, 1, 0} で表される）周りに 45 度回転させます。

lrotate(2, {1, 0, 0}, -30): ID が 2 のオブジェクトを、自身のローカル X 軸（ベクトル {1, 0, 0} で表される）周りに -30 度（時計回り）回転させます。

lrotate("SphereA", normalize({1, 1, 1}), 90): 名前が "SphereA" のオブジェクトを、ベクトル {1, 1, 1} を正規化（長さが 1 になるように調整）した軸周りに 90 度回転させます。正規化することで、回転軸の方向のみが指定され、回転のスケールに影響を与えません。

make_fly_eye_lens

アクション：

指定された要素を複数回繰り返してレンズアレイを作成します

構文：

```
make_fly_eye_lens(nm, ele, size_x, num_x, size_y, num_y)
```

引数：

nm 文字列（作成されるオブジェクトの名前）

ele 文字列 | 整数（レンズアレイ作成の基本要素）

size_x 実数（基本要素コンポーネントあたりの x 方向のサイズ）

num_x 整数（x 方向の繰り返し数）

size_y 実数（基本要素コンポーネントあたりの y 方向のサイズ）

num_y 整数（y 方向の繰り返し数）

戻り値：

null

説明：

TracePro で、指定された要素を複数回繰り返してレンズアレイを作成します。

makelensarray(), make_lens_array(), makeflyeyelens() と同等です。

既存のレンズ要素などを指定されたサイズと繰り返し数でグリッド状に配置し、フライアイレンズのような構造を効率的に作成できます。

make_hex_lens_array

アクション:

指定された要素を複数回繰り返すことで、六角形の形状のレンズアレイを作成します」と説明されています。

構文:

```
make_hex_lens_array(nm, ele, size, num_x, num_y)
```

引数:

nm: 文字列 (string)。作成されるオブジェクト (レンズアレイ) の名前を指定します。

ele: 文字列 (string) または整数 (integer)。レンズアレイを構成する要素を指定します。これは、既存のオブジェクトの名前 (文字列) または要素の種類を示す識別子 (整数) です。

size: 倍精度浮動小数点数 (double)。要素となるコンポーネントのサイズを指定します。これは、レンズの直径やその他の寸法に関連する数値です。

num_x: 整数 (integer)。X 方向 (水平方向) に要素を繰り返す回数を指定します。

num_y: 整数 (integer)。Y 方向 (垂直方向、または六角形配列のもう一つの主要な方向) に要素を繰り返す回数を指定します。

戻り値:

null

説明:

指定された要素を複数回繰り返すことで、六角形の形状のレンズアレイを作成します

例:

`make_hex_lens_array("my_hex_array", "my_lens", 2.5, 5, 4);` "my_lens" という名前の要素 (既存のレンズオブジェクトである可能性が高い) を、サイズ 2.5 で、X 方向に 5 個、Y 方向に 4 個繰り返して、「my_hex_array」という名前の六角形レンズアレイを作成します。

`make_hex_lens_array("LED_grid", 123, 1.0, 10, 8);` ID が 123 の要素 (おそらく特定の種類の LED を表す整数識別子) を、サイズ 1.0 で、X 方向に 10 個、Y 方向に 8 個繰り返して、「LED_grid」という名前の六角形 LED アレイを作成します。

効率的に多数のレンズや光源を規則的な六角形パターンで配置するために使用します。六角形配列は、平面充填率が高いため、光学デバイスや照明システムなどでよく用いられます。

make_lens_array

アクション:

指定された要素を複数回繰り返してレンズアレイを作成します

構文:

```
make_lens_array(nm, ele, size_x, num_x, size_y, num_y)
```

引数:

nm 文字列 (作成されるオブジェクトの名前)

ele 文字列 | 整数 (レンズアレイを作成するための要素)

size_x 実数 (要素コンポーネントあたりの x サイズ)

num_x 整数 (x 方向への繰り返し数)

size_y 実数 (要素コンポーネントあたりの y サイズ)

num_y 整数 (y 方向への繰り返し数)

戻り値：

null

説明：

TracePro で、指定された要素を複数回繰り返してレンズアレイを作成します。
makelensarray(), makeflyeyelens(), make_fly_eye_lens() と同等です。

make_pie

アクション：

パイを作成します

構文：

```
make_pie(nm, r, start, end, thk[, in_r=0.0])
```

引数：

nm 文字列 (作成されるオブジェクトの名前)
r 実数 (パイの半径)
start 実数 (開始角度、度単位)
end 実数 (終了角度、度単位)
thk 実数 (厚さ)
in_r 実数 (内側の半径、省略可能、デフォルト値は 0.0)

戻り値：

null

説明：

TracePro でパイを作成します。

makeellipsoid

アクション：

TracePro 内に楕円体オブジェクトを作成します

構文：

```
makeellipsoid(nm, rx, ry, rz)
```

引数：

nm 文字列 (作成されるレンズの名前)
rx 倍精度浮動小数点数 (x 軸方向の半径)
ry 倍精度浮動小数点数 (y 軸方向の半径)
rz 倍精度浮動小数点数 (z 軸方向の半径)

戻り値：

null

説明：

TracePro 内に楕円体オブジェクトを作成します。

makeflyeyelens

アクション:

指定された要素を複数回繰り返してレンズアレイを作成します

構文:

```
makeflyeyelens(nm, ele, size_x, num_x, size_y, num_y)
```

引数:

nm 文字列 (作成されるオブジェクトの名前)
ele 文字列 | 整数 (レンズアレイを作成するための要素)
size_x 実数 (要素コンポーネントあたりの x サイズ)
num_x 整数 (x 方向への繰り返し数)
size_y 実数 (要素コンポーネントあたりの y サイズ)
num_y 整数 (y 方向への繰り返し数)

戻り値:

null

説明:

TracePro において、指定された要素を複数回繰り返してレンズアレイを作成します。
makelensarray(), make_lens_array(), make_fly_eye_lens() と同等です。

makehexlensarray

アクション:

指定された要素を複数回繰り返して、六角形のレンズアレイを作成します。

構文:

```
makehexlensarray(nm, ele, size, num_x, num_y)
```

引数:

nm 文字列 (作成されるオブジェクトの名前)
ele 文字列 | 整数 (レンズアレイを作成するための要素)
size 実数 (要素となる構成部品のサイズ)
num_x 整数 (x 方向への繰り返し数)
num_y 整数 (y 方向への繰り返し数)

戻り値:

null

説明:

指定された要素を複数回繰り返して、六角形のレンズアレイを作成します。make_hex_lens_array() と同等です。

例:

makehexlensarray("my_hex_array", "my_lens", 5.0, 7, 5): "my_lens" という要素 (既存のレンズオブジェクトの名前) を基本として、要素サイズ 5.0 の六角形レンズアレイ "my_hex_array" を、x 方向に 7 個、y 方向に 5 個の要素数で作成します。

makehexlensarray("small_array", 123, 1.2, 3, 3): ID が 123 の要素を基本として、要素サイズ 1.2 の六角

形レンズアレイ "small_array" を、x 方向に 3 個、y 方向に 3 個の要素数で作成します。

makelens

アクション:

TracePro で 2 つの定義された面を持つソリッドレンズを作成します

構文:

```
makelens(nm,face1,face2,thk,ap_r)
```

引数:

nm 文字列 (作成されるレンズの名前)

face1 面 (レンズの前面)

face2 面 (レンズの背面)

thk 実数 (レンズの厚さ)

ap_r 実数 (開口半径)

戻り値:

文字列

説明:

TracePro で 2 つの定義された面を持つソリッドレンズを作成します。引数 face1 と face2 は、一連のコマンド face:planar(), face:biconic(), face:freeform()... によって作成されている必要があります。

例:

```
front_surface = face:spherical(10) // 球面レンズの前面を作成 (曲率半径 10)
```

```
back_surface = face:spherical(-15) // 球面レンズの背面を作成 (曲率半径 -15)
```

```
lens_name = "MyDoubleConvexLens"
```

```
thickness = 5.0                    // レンズの厚さ (単位はモデルに依存)
```

```
aperture_radius = 3.0              // レンズの有効半径 (単位はモデルに依存)
```

```
lens_object_name = makelens(lens_name, front_surface, back_surface, thickness, aperture_radius)
```

```
print " 作成されたレンズオブジェクトの名前 :", lens_object_name
```

```
// 他の種類のレンズの例
```

```
planar_face = face:planar()
```

```
second_spherical_face = face:spherical(20)
```

```
concave_lens_name = "MyPlanoConcaveLens"
```

```
concave_thickness = 2.5
```

```
concave_aperture = 4.0
```

```
concave_lens_object = makelens(concave_lens_name, planar_face, second_spherical_face, concave_thickness, concave_aperture)
```

```
print " 作成された凹レンズオブジェクトの名前 :", concave_lens_object
```

makelensarray

アクション:

指定された要素を複数回繰り返すことでレンズアレイを作成します

構文:

makelensarray(nm, ele, size_x, num_x, size_y, num_y)

引数：

nm 文字列 (作成されるオブジェクトの名前)
ele 文字列 | 整数 (レンズアレイを作成するための要素)
size_x 実数 (要素コンポーネントあたりの x サイズ)
num_x 整数 (x 方向の繰り返し数)
size_y 実数 (要素コンポーネントあたりの y サイズ)
num_y 整数 (y 方向の繰り返し数)

戻り値：

null

説明：

TracePro で、指定された要素を複数回繰り返すことでレンズアレイを作成します。

make_lens_array(), makeflyeyelens(), make_fly_eye_lens() と同等です。

指定した基本要素を指定されたサイズと繰り返し数で x 方向と y 方向に配列化し、レンズアレイという光学部品を作成します。

move

アクション：

指定されたオブジェクトを移動します

構文：

move(obj, vec, [dist])

引数：

obj 文字列 | 整数 (移動するオブジェクト)
vec ベクトル (移動方向)
dist 実数 (移動距離)

戻り値：

null

説明：

指定されたオブジェクトを移動します。dist 引数が指定されていない場合、対象オブジェクトは vec の量だけ移動します。

object:export:

アクション：

オブジェクトを TracePro にエクスポートします

構文：

object:export(objID|objName,[name])

引数：

objID: 整数 (integer) で、インタラクティブオブティマイザにおけるオブジェクトの ID です。

objName: 文字列 (string) で、インタラクティブオブティマイザで割り当てられたオブジェクトの名前です。

[name]: 省略可能な引数で、エクスポートするオブジェクトに付けたい名前を表す文字列です。

戻り値:

真偽値

説明:

オブジェクトを TracePro にエクスポートします

インタラクティブオプティマイザで選択したオブジェクトを、ID または名前で指定し、必要に応じて新しい名前を付けて TracePro に送ることができます。

or

アクション:

2 つのオブジェクトに対してユニオン（和）のブール演算を実行します

構文:

or(obj1,obj2)

引数:

obj1 文字列 | 整数 (ユニオン演算の最初のオブジェクト)

obj2 文字列 | 整数 (ユニオン演算の 2 番目のオブジェクト)

戻り値:

null

説明:

2 つのオブジェクトに対してユニオン（和）のブール演算を実行します。オブジェクトを指定するには、整数または文字列を使用できます。整数はインタラクティブオプティマイザーで割り当てられたオブジェクト ID を意味し、文字列は TracePro で認識される名前です。unite() と同等です。

ray_test:

アクション:

テスト光線を用いてオブジェクト上の交点の位置を取得するためのレイテストを実行します

構文:

ray_test(obj, ray_o, ray_d)

引数:

obj: テスト対象のオブジェクトを指定します。オブジェクトの名前 (文字列) またはオブジェクトの ID (整数) のいずれかを指定できます。

ray_o: テスト光線の開始位置 (position) を示す 3 次元ベクトルです。

ray_d: テスト光線の方向 (vector) を示す 3 次元ベクトルです。通常は正規化された方向ベクトルが使用されます。

戻り値:

position 光線が交差した場合、その交点の 3 次元位置ベクトルが返されます。光線がオブジェクトと交差しない場合、無効な位置を示す値 (例えば null や特定の範囲外の値) が返される可能性があります。

説明:

テスト光線を用いてオブジェクト上の交点の位置を取得するためのレイテストを実行します。

ある点 (ray_o) からある方向 (ray_d) に発射された仮想的な光線が、指定されたオブジェクト (obj) とどこで交わるかを計算します。

例：

```
// オブジェクト名でレイテストを実行
intersection_point1 = ray_test("target_object", {0, 0, 0}, {0, 1, 0})

// オブジェクト ID でレイテストを実行
object_id = get_object_id("another_object")
intersection_point2 = ray_test(object_id, {1, 1, 1}, normalize({-1, -1, -1}))

// レイがオブジェクトに当たらなかった場合、無効な位置 (例: null や特定の範囲の値) が返される可能性があります
if (isValidPosition(intersection_point1)) then
  print " オブジェクト 'target_object' との交点 :", intersection_point1
else
  print " オブジェクト 'target_object' との交点なし "
endif

if (isValidPosition(intersection_point2)) then
  print " オブジェクト ID " + cstr(object_id) + " との交点 :", intersection_point2
else
  print " オブジェクト ID " + cstr(object_id) + " との交点なし "
endif
```

remove

アクション：

オブジェクト A と B の間の重複部分をオブジェクト A から削除します

構文：

```
remove(objA, objB)
```

引数：

objA: 操作の対象となるオブジェクト A を指定します。文字列（オブジェクト名）または整数（オブジェクト ID）を使用できます。

objB: 重複部分を削除するために使用するオブジェクト B を指定します。文字列（オブジェクト名）または整数（オブジェクト ID）を使用できます。

戻り値：

null

説明：

オブジェクト A と B の間の重複部分をオブジェクト A から削除します。オブジェクトを指定するには、整数または文字列のいずれかを使用できます。整数は Interactive Optimizer で割り当てられたオブジェクト ID を意味し、文字列は TracePro で認識される名前です。このコマンドと subtract() の違いは、subtract() は操作完了時にオブジェクト B を削除しますが、remove() は削除しないことです。」とあります。

例：

```
remove("Part1", "Cylinder001") // TracePro の名前でオブジェクトを指定
remove(15, 22)                 // Interactive Optimizer のオブジェクト ID で指定
```

scale

アクション:

指定された拡大率と拡大位置でボディを拡大縮小します

構文:

```
scale(obj, scale_x, scale_y, scale_z, scale_pos, fix)
```

引数:

obj 文字列 | 整数 (オブジェクトの名前または ID)

scale_x 倍精度浮動小数点数 (x 軸方向の拡大率)

scale_y 倍精度浮動小数点数 (y 軸方向の拡大率)

scale_z 倍精度浮動小数点数 (z 軸方向の拡大率)

scale_pos 真偽値 (位置を拡大縮小するかどうか)

fix 真偽値 (変換を固定するかどうか)

戻り値:

null

説明:

scale-position フラグは、位置を拡大縮小することを許可します。fix-transform フラグは、変換をボディに固定することを許可します。copy フラグは、元のエンティティのコピーを作成することを許可します。

seet:skin_wires

アクション:

一連のワイヤを補間するシートボディを作成します

構文:

```
sheet:skin_wires(wb, nm)
```

引数:

wb 文字列のリスト (ワイヤボディ名のリスト)

nm 文字列 (作成されるシートの名前)

戻り値:

null

説明:

指定されたワイヤは、作成されるシートボディによって補間される断面を定義します。少なくとも 2 つのワイヤボディが必要です。ワイヤボディは単純であると見なされます。つまり、各ボディの最初のワイヤのみがスキニングに使用されます。ワイヤは開いていても閉じていてもかまいません。ワイヤはコピーされます。つまり、元のワイヤは残ります。ワイヤは端点を共有でき、C1 連続接線である必要はありません。

sheet:skin_wires_path

アクション:

一連のワイヤーを補間し、パスにも沿うシートボディを作成します

構文:

sheet:skin_wires_path(wb, path, nm)

引数:

wb: 文字列のリスト (list of string) で、ワイヤーボディの名前のリストです。補間のための断面形状を定義するワイヤーボディの名前を複数指定します。

path: 文字列 (string) で、パスの名前です。作成されるシートボディが沿う経路を指定します。

nm: 文字列 (string) で、結果として作成されるシートボディの名前です。

戻り値:

null

説明:

指定されたワイヤーは、結果として作成されるシートボディによって補間される断面を定義します。少なくとも 2 つのワイヤーボディが必要です。

ワイヤーボディは単純であると想定されます。つまり、各ボディの最初のワイヤーのみがスキニングに使用されます。(スキニングとは、複数の断面形状を滑らかに繋いでサーフェスを作成する手法です) ワイヤーは開いているか閉じているかのどちらでもかまいません。

ワイヤーはコピーされます。つまり、元のワイヤーは残ります。

ワイヤーは端点を共有でき、C1 連続接線である必要はありません。(C1 連続接線とは、形状が滑らかに繋がっている状態を指しますが、ここでは必ずしもその必要がないということです)

例:

提示された例 sheet:skin_wires_path(wire_bodies, "spline_path", "skinned_sheet") は、以下の処理を行うことを示唆しています。

wire_bodies: という名前の文字列リストで、補間したいワイヤーボディの名前が格納されています。例えば、{"wire1", "wire2", "wire3"} のような形式です。

"spline_path": という名前のパス(おそらくあらかじめ定義されているスプライン曲線など)に沿って、シートボディが作成されます。

"skinned_sheet": 作成されたシートボディに "skinned_sheet" という名前が付けられます。

slinterface:skinning

アクション:

ワイヤーボディのリストからスキニングオブジェクトを作成します

構文:

slinterface:skinning(wb, nm)

引数:

wb 文字列のリスト (ワイヤーボディ名のリスト)

nm 文字列 (作成されるオブジェクトの名前)

戻り値:

null

説明:

ワイヤーボディのリストからスキニングオブジェクトを作成します。これは、slinterface を定義することでスキニングオブジェクトを作成するためのスキームコマンド実行のショートカットです。スキニングに使用するワイヤーボディは、この関数を実行する前に作成 / 定義されている必要があります。

解説：

この関数 `slinterface:skinning` は、3D グラフィックスやアニメーションの分野で用いられる「スキニング」という処理に関連しています。具体的には、複数の「ワイヤーボディ」と呼ばれる骨格のような構造に基づいて、滑らかな表面を持つ「スキニングオブジェクト」を作成するために使用されます。

subtract

アクション：

2 つのオブジェクトに対して差分ブール演算を実行します

構文：

`ubtract(obj1, obj2)`

引数：

`obj1`: 差分演算の対象となる最初のオブジェクトを指定します。文字列または整数のいずれかの型を使用できます。

`obj2`: `obj1` から取り除くオブジェクトを指定します。`obj1` と同様に、文字列または整数のいずれかの型を使用できます。

戻り値：

`null`

説明：

2 つのオブジェクトに対して差分ブール演算を実行します。オブジェクトを指定するには、整数または文字列のいずれかを使用できます。整数は Interactive Optimizer で割り当てられたオブジェクト ID を意味し、文字列は TracePro で認識される名前です。

例：

`subtract("PartA", "CutoutB")`: 「PartA」という名前のオブジェクトから、「CutoutB」という名前のオブジェクトが重なっている部分を取り除きます。

`subtract(15, 22)`: ID が 15 のオブジェクトから、ID が 22 のオブジェクトが重なっている部分を取り除きます。

surface:domain

アクション：

指定された面に対し、その基となるサーフェスの定義域を返します。

構文：

`surface:domain(face)`

引数：

`face` 文字列 | 面 (面の名前、または `face()` によって作成された面オブジェクト)

戻り値：

リスト (`uMin`, `uMax`, `vMin`, `vMax`)

説明：

指定された面に対し、その基となるサーフェスの定義域を返します。

surface:eval_normal

アクション：

法線ベクトルを評価します。

構文：

surface:eval_normal(f,u,v) | surface:eval_normal(f,pos)

引数：

f 文字列 | 面 (面の名前、または face() で作成された面オブジェクト)

u 実数 (u パラメータ値)

v 実数 (v パラメータ値)

pos 位置

戻り値：

ベクトル

説明：

この拡張機能は、与えられたパラメータ値を持つパラメトリックサーフェス上の法線ベクトルを見つけます。

surface:eval_pos

アクション：

位置を評価します。

構文：

surface:eval_pos(f, u, v)

引数：

f 文字列 | 面 (面の名前、または face() で作成された面オブジェクト)

u 実数 (u パラメータ値)

v 実数 (v パラメータ値)

戻り値：

pos

説明：

この拡張機能は、指定されたパラメータ値を持つパラメトリックサーフェス上の位置を見つけます。

surface:eval_uv:

アクション：

サーフェス上の uv を評価します

構文：

surface:eval_uv(f, pos)

引数：

f 文字列 (面の名前) または face オブジェクト (face() 関数などで作成された面オブジェクト) です。

UV 座標を評価したい対象のサーフェスを指定します。

pos: position (位置) です。サーフェス上のどの位置に対応する UV 座標を知りたいかを指定します。通常は 3D 空間の座標 {x, y, z} のような形式で表されます。

戻り値:

2 つの double を持つリスト。最初の要素が u、2 番目の要素が v です。指定されたサーフェス上の位置に対応する u 値と v 値を格納したリストを返します。

説明:

このエクステンションは、与えられた位置を持つパラメトリックサーフェス上の uv を見つけます

例:

```
// 例: 名前が "my_surface" の面上の位置 {1, 2, 3} に対応する UV 座標を取得
uv_coords = surface:eval_uv("my_surface", {1, 2, 3})
print uv_coords // 結果は例えば {0.5, 0.7} のようなリストになります

// 例: 変数 myFace に格納された面オブジェクト上の位置 {4, 5, 6} に対応する UV 座標を取得
uv_coords_from_face = surface:eval_uv(myFace, {4, 5, 6})
print uv_coords_from_face
```

surface:get_coordinate

アクション:

指定された UV 座標と UV 座標における上方向から座標を取得します。

構文:

```
surface:get_coordinate(f, u, v, up_uv)
```

引数:

f 文字列 | 面 (面の名前または face() で作成された面オブジェクト)
u 実数 (U パラメータ値)
v 実数 (V パラメータ値)
up_uv 2D 位置

戻り値:

座標構造体

説明:

指定された UV 座標と UV 座標における Up 方向から座標を取得します。

surface:get_u_dir

アクション:

指定された uv 位置における u 方向を評価します

構文:

```
surface:get_u_dir(f, u, v)
```

引数:

f 文字列 | 面 (面の名前、または face() で作成された面オブジェクト)
u 実数 (u パラメータの値)
v 実数 (v パラメータの値)

戻り値：

ベクトル

説明：

指定された uv 位置における u 方向を評価します。

surface:get_v_dir

アクション：

指定された uv 位置における v 方向を評価します

構文：

surface:get_v_dir(f, u, v)

引数：

f 文字列 | 面 (面の名前、または face() で作成された面オブジェクト)

u 実数 (u パラメータの値)

v 実数 (v パラメータの値)

戻り値：

ベクトル

説明：

指定された uv 位置における v 方向を評価します。

sweep:vector

アクション：

指定されたベクトルに沿って面をスイープします

構文：

sweep:vector(f, vec)

引数：

f: スイープする面 (face) を指定します。文字列型 (面の名前) または面オブジェクトそのもので指定できるようです。

vec: スイープの方向と距離を表すベクトル (vector) です。通常、{x, y, z} のような形式で記述されます。ベクトルの成分が移動量を示し、ベクトルの方向が移動方向を示します。

戻り値：

null

説明：

指定されたベクトルに沿って面をスイープします

例：

sweep:vector("Face1", {10, 0, 0}): 名前が "Face1" の面を、X 軸方向に 10 単位スイープします。Y 軸と Z 軸方向への移動はありません。

sweep:vector(getSelectedFace(), {0, 5, 2}): 現在選択されている面を取得し、それをベクトル {0, 5, 2} に沿ってスイープします。これは、X 軸方向には移動せず、Y 軸方向に 5 単位、Z 軸方向に 2 単位移動しながらスイープすることを意味します。getSelectedFace() は、選択された面オブジェクトを返す別の関

数であると推測されます。

sweep:vector("CylinderTop", normalize({1, 1, 1}) * 3): 名前が "CylinderTop" の面を、ベクトル {1, 1, 1} の方向に 3 単位スイープします。normalize() 関数はベクトルを正規化（長さ 1 のベクトルにする）する関数であると推測され、それに * 3 を掛けることで、方向は {1, 1, 1} のままで長さが 3 のスイープベクトルを作成しています。

tp:edge:

アクション:

定義された辺を TracePro にエクスポートします

構文:

```
tp:edge(nm, edge)
```

引数:

nm: 文字列 (string) です。TracePro 内で作成される辺に付ける名前を指定します。これにより、TracePro 上でエクスポートされた辺を識別できます。

edge: 「edge」オブジェクトです。これは、おそらく edge() という別のコマンドによって作成された、エクスポートしたい辺の情報を保持するオブジェクトです。このオブジェクトの具体的な構造はここでは不明です。

戻り値:

null

説明:

定義された辺を TracePro にエクスポートします

例:

```
tp:edge("MyEdge", myEdgeObject)
```

上記の構文 tp:edge("MyEdge", myEdgeObject) は、myEdgeObject という辺オブジェクトを、TracePro 内で "MyEdge" という名前でエクスポートする処理を示唆しています。

tp:wire_body

アクション:

定義されたエッジをワイヤーボディとして TracePro にエクスポートします

構文:

```
tp:wire_body(nm, edge)
```

引数:

nm 文字列 (TracePro で作成されるエッジの名前)

edge エッジ (edge() コマンドで作成されたエッジオブジェクト)

戻り値:

null

説明:

定義されたエッジをワイヤーボディとして TracePro にエクスポートします

unite

アクション:

2つのオブジェクトに対して結合（和）のブール演算を実行します

構文:

```
unite(obj1, obj2)
```

引数:

obj1 文字列 | 整数 (結合演算の 1 番目のオブジェクト)

obj2 文字列 | 整数 (結合演算の 2 番目のオブジェクト)

戻り値:

null

説明:

2つのオブジェクトに対して結合（和）のブール演算を実行します。オブジェクトを指定するには、整数または文字列のいずれかを使用できます。整数はインタラクティブオブティマイザーで割り当てられたオブジェクト ID を意味し、文字列は TracePro で認識される名前です。or() と同等です。

トランスフォーム関数

trans:transform

アクション:

入力された位置 / ベクトルに変換行列を適用して、変換後のものを取得します

構文:

```
trans:transform(mx, pos|vec)
```

引数:

mx 変換 (入力位置に適用する変換行列)

pos 位置 (変換したい位置)

vec ベクトル (変換したいベクトル)

戻り値:

位置

説明:

変換行列から変換された位置を返します。transform:transform() と同等です。

trans:transform_norm

アクション:

入力されたベクトル / 位置に変換行列を適用し、並進成分を除いた変換後のものを取得します

構文:

```
trans:transform_norm(mx, pos|vec)
```

引数:

mx 変換 (入力位置に適用する変換行列)

pos 位置 (変換したい位置)

vec ベクトル (変換したいベクトル)

戻り値:

位置 / ベクトル

説明:

変換行列から変換された位置 / ベクトルを返します。transform:transform_norm() と同等です。

transform:transform() との違いは、これは入力ベクトルの方向のみを変換し、並進は無視される点です。

向きの変化を計算したい場合は、transform:transform_norm() を使用するのが適切なコマンドです。

trans_compose

アクション:

2 つの変換行列を合成した結果の行列を返します

構文:

```
trans_compose(m1, m2)
```

引数:

m1 変換行列

m2 変換行列

戻り値：

変換行列

説明：

2 つの変換行列を合成した結果の行列を返します。transform:compose() と同等です。

trans_identity

アクション：

恒等変換行列を返します

構文：

trans_identity()

引数：

null

戻り値：

変換 (transform)

説明：

恒等変換行列を返します。transform:identity() と等価です。

trans_inverse

アクション：

入力された行列の逆行列を返します

構文：

trans_inverse(mx)

引数：

mx 変換行列 (transform)

戻り値：

変換行列 (transform)

説明：

入力された行列の逆行列を返します。transform:inverse() と同等です。

original_transform = translate(10, 0, 0): X 方向に 10 単位移動させる変換行列を作成します。

inverse_transform = trans_inverse(original_transform): original_transform の逆行列を計算します。これは、X 方向に -10 単位移動させる変換行列になるはずです。

point = {5, 0, 0}: 元の点の座標です。

transformed_point = transform_point(original_transform, point): 元の点に original_transform を適用し、点を移動させます。

original_point_again = transform_point(inverse_transform, transformed_point): 移動後の点に inverse_transform を適用すると、元の点の座標に戻るはずです。

同様に、より複雑な変換（移動と回転の組み合わせなど）に対しても、その逆変換を trans_inverse で計

算し、適用することで元の状態に戻すことができます。

この関数は、3D 空間でのオブジェクトの操作や、座標系の変換などを扱う際に非常に役立ちます。

trans_mapping

アクション:

元の座標を新しい目的の座標に完全にマッピングできる変換行列を返します

構文:

```
trans_mapping(fromCen, fromNorm, fromUp, toCen, toNorm, toUp)
```

引数:

fromCen 位置 (元の座標の中心)

fromNorm ベクトル (元の座標の法線)

fromUp ベクトル (元の座標の上方向)

toCen 位置 (目的の座標の中心)

toNorm ベクトル (目的の座標の法線)

toUp ベクトル (目的の座標の上方向)

戻り値:

変換 (transform)

説明:

元の座標を新しい目的の座標に完全にマッピングできる変換行列を返します。transform.mapping() と同等です。

trans_mapping 関数は、以下の処理を行うことで変換行列を生成します。

1. 回転: 元の座標系の方向ベクトル (fromNorm, fromUp) を、目的の座標系の対応する方向ベクトル (toNorm, toUp) に一致させるための回転を計算します。これには、座標軸の向きを合わせるための回転が含まれます。
2. 並進: 元の座標系の中心 (fromCen) を、目的の座標系の中心 (toCen) に移動させるための並進ベクトルを計算します。

```
original_center = {0, 0, 0}
```

```
original_normal = {0, 0, 1}
```

```
original_up = {0, 1, 0}
```

```
destination_center = {10, 5, 2}
```

```
destination_normal = {1, 0, 0}
```

```
destination_up = {0, 1, 0}
```

```
mapping_matrix = trans_mapping(original_center, original_normal, original_up, destination_center,  
destination_normal, destination_up)
```

```
// 別の例
```

```
from_center = {1, 1, 1}
```

```
from_normal = normalize({1, 1, 1})
```

```
from_up = normalize({-1, 1, 0})
```

```
to_center = {-2, 3, 0}
```

```
to_normal = {0, -1, 0}
```

```
to_up = {0, 0, 1}
```

```
transform_matrix = trans_mapping(from_center, from_normal, from_up, to_center, to_normal, to_up)
```

trans_rotatex

アクション:

X 軸周りの回転操作のための回転変換行列を返します

構文:

```
trans_rotatex(deg)
```

引数:

deg 倍精度浮動小数点数 (double)

戻り値:

変換 (transform)

説明:

X 軸周りの回転操作のための回転変換行列を返します。

transform.rotatex() と等価です。

trans_rotatey

アクション:

y 軸周りの回転操作のための回転変換行列を返します

構文:

```
trans_rotatey(deg)
```

引数:

deg double (倍精度浮動小数点数)

戻り値:

transform (変換行列)

説明:

y 軸周りの回転操作のための回転変換行列を返します。

transform.rotatey() と等価です。

trans_rotatez()

アクション:

z 軸周りの回転操作のための回転変換行列を返します

構文:

```
trans_rotatez(deg)
```

引数:

deg: 「double」型であり、z 軸周りの回転角度を度単位で指定します。

戻り値:

transform」型であり、z 軸周りの指定された角度の回転を表す変換行列を返します。

説明：

z 軸周りの回転操作のための回転変換行列を返します。transform:rotatez() と等価です
transform:rotatez() という別の関数またはメソッドと同じ機能を持ちます。

どのように使用するか：

trans_rotatez() 関数は、3D 空間内のオブジェクトや座標系を z 軸を中心に回転させるために使用される変換行列を生成します。この行列は、通常、ベクトルや点の座標に適用することで、回転後の新しい座標を得ることができます。

例：

rotation_matrix_z = trans_rotatez(45.0): z 軸周りに 45 度回転させるための変換行列を rotation_matrix_z 変数に格納します。

rotation_matrix_z2 = trans_rotatez(90.0): z 軸周りに 90 度回転させるための変換行列を rotation_matrix_z2 変数に格納します。

rotation_matrix_z_negative = trans_rotatez(-30.0): z 軸周りに -30 度（時計回り）回転させるための変換行列を rotation_matrix_z_negative 変数に格納します。

trans_scale

アクション：

指定されたスケーリング係数を持つスケーリング行列を返します

構文：

```
trans_scale(scale|[sx,sy,sz])
```

引数：

scale 倍精度浮動小数点数 (全軸に対する均一なスケーリング係数)

sx 倍精度浮動小数点数 (x 軸方向のスケーリング係数)

sy 倍精度浮動小数点数 (y 軸方向のスケーリング係数)

sz 倍精度浮動小数点数 (z 軸方向のスケーリング係数)

戻り値：

変換 (transform)

説明：

指定されたスケーリング係数を持つスケーリング行列を返します。transform:scale() と同等です。

trans_translate

アクション：

並進変換行列を返します

構文：

```
trans_translate([dx,dy,dz]|vec)
```

引数：

dx 実数

dy 実数

dz 実数

vec ベクトル

戻り値：

変換

説明：

指定されたベクトルによる並進操作を行う変換行列を返します。

`transform:translate()` と等価です。

transform:compose

アクション：

2 つの変換行列を合成した結果の行列を返します

構文：

`transform:compose(m1, m2)`

引数：

m1 変換行列

m2 変換行列

戻り値：

変換行列

説明：

2 つの変換行列を合成した結果の行列を返します。`trans_compose()` と同等です。

例：

`transform1 = translate({1, 2, 3})`

`transform2 = rotate({0, 0, 1}, 45)`

`composed_transform = transform:compose(transform1, transform2)`

最初の例では、`transform1` はオブジェクトを (1, 2, 3) だけ平行移動する変換行列を表し、`transform2` は Z 軸を中心に 45 度回転させる変換行列を表しています。`transform:compose` を使用すると、まず回転が行われ、次に平行移動が行われる効果を持つ `composed_transform` が作成されます。

`transform_a = scale({2, 2, 2})`

`transform_b = translate({-1, 0, 0})`

`composed_transform_2 = transform:compose(transform_a, transform_b)`

2 番目の例では、`transform_a` はオブジェクトを各軸に沿って 2 倍に拡大縮小する変換行列を表し、`transform_b` はオブジェクトを X 軸の負の方向に 1 だけ平行移動する変換行列を表しています。`transform:compose` を使用すると、まず平行移動が行われ、次に拡大縮小が行われる効果を持つ `composed_transform_2` が作成されます。

transform:identity

アクション：

恒等変換行列を返します

構文：

`transform:identity()`

引数：

null

戻り値：

transform

説明：

恒等変換行列を返します。trans_identity() と等価です

恒等変換行列（Identity Transform Matrix）とは：

数学やコンピュータグラフィックスにおいて、恒等変換行列は、オブジェクトに対して何の効果も与えない特別な行列です。これは、オブジェクトの位置、回転、スケールなどを一切変更しません。行列演算において、恒等行列を他の行列に掛けても、元の行列がそのまま変わらないという性質を持ちます。

通常、3次元空間における 4x4 の恒等変換行列は以下のようになります。

[1 0 0 0]

[0 1 0 0]

[0 0 1 0]

[0 0 0 1]

この関数は、このような恒等変換行列を生成して返す役割を持ちます。これは、初期状態や基準となる変換がない状態を表す際に役立ちます。例えば、オブジェクトの初期位置を設定したり、既存の変換をリセットしたりする場合などに使用されます。

transform:inverse

アクション：

入力行列の逆行列を返します

構文：

transform:inverse(mx)

引数：

mx 変換 (transform)

戻り値：

変換 (transform)

説明：

入力行列の逆行列を返します。trans_inverse() と同等です。

逆行列について：

変換行列の逆行列は、その変換を打ち消す（元に戻す）効果を持つ行列です。例えば、あるオブジェクトを特定の方向に移動させる変換行列の逆行列は、そのオブジェクトを元の位置に戻す移動を行います。回転や拡大縮小についても同様です。

逆行列は、3D グラフィックス、ロボティクス、物理シミュレーションなど、さまざまな分野でオブジェクトの操作や座標系の変換を扱う際に重要な概念です。

この関数を使用することで、与えられた変換行列 mx に対して、その逆変換を表す行列を取得できることがわかります。

transform:mapping

アクション:

元の座標を新しい目的地に完全にマッピングできる変換行列を返します

構文:

```
transform:mapping([fromCen, fromNorm, fromUp, toCen, toNorm, toUp]|[fromCoord, toCoord])
```

引数:

fromCen position (元の座標の中心)

fromNorm vector (元の座標の法線)

fromUp vector (元の座標の上方向)

toCen position (目的地の座標の中心)

toNorm vector (目的地の座標の法線)

toUp vector (目的地の座標の上方向)

fromCoord coordinate (元の座標)

toCoord coordinate (目的地の座標)

戻り値:

transform

説明:

元の座標を新しい目的地に完全にマッピングできる変換行列を返します。trans_mapping() と同等です。

2つの座標系の間の相対的な位置と向きに基づいて、一方の座標系からもう一方の座標系への変換を行うための変換行列を計算する関数です。引数の指定方法が2通り用意されており、個々のベクトルを指定する方法と、座標系オブジェクト全体を指定する方法があります。

transform:rotate

アクション:

指定された方向に沿って回転させる操作のための回転変換行列を返します。

構文:

```
transform:rotate([axis,deg]|[fromV,toV])
```

1つ目は、回転軸を表すベクトル (axis) と、回転角度 (度単位) を表す double 型の値 (deg) のペアです。

2つ目は、回転前のベクトル (fromV) と回転後のベクトル (toV) のペアです。

引数:

axis: ベクトル (回転軸)。回転の中心となる方向を指定します。

deg: double (回転角度、度単位)。指定された軸周りの回転量を度数で指定します。

fromV: ベクトル (回転前のベクトル)。

toV: ベクトル (回転後のベクトル)。

戻り値:

transform (変換行列)。回転操作を表す変換行列が返されます。この行列をベクトルや点に乗算することで、回転後の新しいベクトルや点の位置を得ることができます。

説明:

指定された軸に沿って回転させる操作のための回転変換行列を返します。または、回転前のベクトルと回転後のベクトルを入力すると、回転前のベクトルを回転後のベクトルに変換できる回転行列が返され

ます。trans_rotate() と同じ機能です。

補足：

変換行列 (Transform Matrix): 3D 空間における回転、平行移動、拡大縮小などの幾何学的変換を数学的に表現する行列です。ベクトルや点にこの行列を乗算することで、変換後の座標を計算できます。

回転軸 (Rotation Axis): 回転の中心となる直線の方角を示すベクトルです。

回転角度 (Rotation Angle): 指定された軸周りの回転量を示す角度です (通常は度単位またはラジアン単位で指定します)。

from ベクトル (From Vector) と to ベクトル (To Vector): これらのベクトルを指定すると、関数は fromV を toV に重ね合わせるような回転を行う変換行列を計算します。

同等の関数 (Equivalent to trans_rotate()): transform:rotate と trans_rotate() は同じ機能を持つ別の名前の関数であることを示しています。

この関数は、3D グラフィックス、ロボティクス、物理シミュレーションなどの分野で、オブジェクトを特定の軸周りに回転させたり、ある方角を別の方角に合わせるための回転を計算したりする際に非常に役立ちます。

例：

// 例 1: Z 軸を中心に 30 度回転させる変換行列を作成

```
rotationMatrix1 = transform:rotate({0, 0, 1}, 30)
```

// 例 2: ベクトル {1, 0, 0} をベクトル {0, 1, 0} に変換する回転行列を作成

```
rotationMatrix2 = transform:rotate({1, 0, 0}, {0, 1, 0})
```

// 例 3: Y 軸を中心に -45 度回転させる変換行列を作成

```
rotationMatrix3 = transform:rotate({0, 1, 0}, -45)
```

// 例 4: ベクトル {0, 0, 1} をベクトル {1, 0, 0} に変換する回転行列を作成

```
rotationMatrix4 = transform:rotate({0, 0, 1}, {1, 0, 0})
```

transform:rotatex

アクション：

x 軸を中心に回転させる操作のための回転変換行列を返します

構文：

```
transform:rotatex(deg)
```

引数：

deg double (倍精度浮動小数点数)

戻り値 : transform (変換行列)

説明：

x 軸を中心に回転させる操作のための回転変換行列を返します。trans_rotatex() と等価です。

指定された角度 (度数法) で、3 次元空間内のオブジェクトを x 軸を中心に回転させるための変換行列を生成して返します。この行列をオブジェクトの座標に適用することで、実際に回転が行われます。

transform:rotatey

アクション：

y 軸周りの回転操作のための回転変換行列を返します

構文：

transform:rotatey(deg)

引数：

deg double (角度)

戻り値：

transform (変換行列)

説明：

y 軸周りの回転操作のための回転変換行列を返します。trans_rotatey() と同等です。

transform:rotatez

アクション：

z 軸周りの回転操作のための回転変換行列を返します

構文：

transform:rotatez(deg)

引数：

deg double (倍精度浮動小数点数)

戻り値：

transform (変換行列)

説明：

z 軸周りの回転操作のための回転変換行列を返します。trans_rotatez() と同等です。

指定された角度(deg)だけz軸を中心に回転させるための3x3または4x4の変換行列を生成する関数です。
この行列を3Dオブジェクトの頂点座標などに適用することで、オブジェクトをz軸周りに回転させることができます。

transform:scale

アクション：

指定されたスケーリング係数を持つスケーリング行列を返します

構文：

transform:scale(scale|[sx,sy,sz])

引数：

scale 倍精度浮動小数点数

sx 倍精度浮動小数点数 (x 軸方向のスケーリング係数)

sy 倍精度浮動小数点数 (y 軸方向のスケーリング係数)

sz 倍精度浮動小数点数 (z 軸方向のスケーリング係数)

戻り値：

transform (変換行列)

説明：

指定されたスケーリング係数を持つスケーリング行列を返します。trans_scale() と等価です。

オブジェクトを指定された倍率で拡大または縮小するための変換行列を生成するツールです。均一スケー

リングと非均一スケーリングの両方に対応しています。

transform:transform

アクション:

入力された位置 / ベクトルに変換行列を適用し、変換後の結果を取得します

構文:

```
trans:transform(mx, pos|vec)
```

引数:

mx transform (入力位置に適用する変換行列)

pos position (変換したい位置)

vec vector (変換したいベクトル)

戻り値:

position

説明:

変換行列から変換された位置を返します。trans:transform() と同等です。

与えられた変換行列を使って、指定された位置やベクトルを別の位置に変換する関数です。

transform:transform_norm

アクション:

変換行列を入力ベクトル / 位置に適用し、平行移動なしで変換されたものを取得します

構文:

```
transform:transform_norm(mx, pos|vec)
```

引数:

mx tranform (入力位置に適用する変換行列)

pos position (変換したい位置)

vec vector (変換したいベクトル)

戻り値:

position/vector

説明:

変換行列から変換された位置 / ベクトルを返します。trans:transform_norm() と同等です。

transform:transform() との違いは、これは入力ベクトルの方向のみを変換し、平行移動は無視される点です。

向きの変化を計算したい場合は、transform:transform_norm() が適切なコマンドです。

与えられた変換行列を用いて、位置やベクトルの向きを変換する際に、行列の平行移動成分を無視して回転や拡大縮小のみを適用する関数です。オブジェクトの向きの変化や、ベクトルが変換によってどのように回転するかを計算するのに役立ちます。

transform:translate

アクション:

並進変換行列を返します

構文:

```
transform:translate([dx,dy,dz]|vec)
```

引数:

dx 倍精度浮動小数点数

dy 倍精度浮動小数点数

dz 倍精度浮動小数点数

vec ベクトル

戻り値:

変換

説明:

指定されたベクトルによる並進操作を行う変換行列を返します。

trans_translate() と等価です。

3D 空間内のオブジェクトを指定した量だけ平行移動させるための変換行列を生成する関数であり、移動量を x, y, z の各軸方向の距離として個別に指定するか、それらをまとめたベクトルとして指定することができます。この行列をオブジェクトの頂点座標などに適用することで、オブジェクトを目的の位置へ移動させることができます。

