基本几何造型函数

//立方体
api_make_cuboid(length(x),width(y),height(z),BODY)
api_solid_block(SPAposition(左上角顶点坐标), SPAposition(右下角顶点坐标),BODY)

//球体
api_make_sphere(半径，BODY)
api_solid_sphere(SPAposition(圆心坐标),半径,BODY)

//圆环体
api_make_torus(外环半径,环宽,BODY)
api_solid_torus(SPAposition(环心坐标), 外环半径,环宽,BODY)

//圆锥体
api_make_frustum(height(z), length(x),width(y),顶部半径,BODY)
api_solid_cylinder_cone(SPAposition(顶部圆心坐标), SPAposition(底部圆心坐标),m*M_PI(底部长轴),n*M_PI(底部短轴),0(顶部半径),NULL,BODY)

//圆柱体
api_make_frustum(height(z), length(x),width(y),顶部半径,BODY)
api_solid_cylinder_cone(SPAposition(顶部圆心坐标), SPAposition(底部圆心坐标),m*M_PI(底部长轴),n*M_PI(底部短轴),0(顶部半径),NULL,BODY)

//棱柱
api_make_prism(height(z), length(x),width(y),棱数,BODY)

//棱锥
api_make_pyramid(height(z),length(x),width(y),顶部半径,棱数,BODY)

//闭合样条
api_curve_spline(int,SPAposition*,SPAunit_vector*(NULL),SPAunit_vector*(NULL),EDGE*&,logical(TRUE),logical(TRUE),AcisOptions*)

//曲线
api_curve_line(const SPAposition& pt1, const SPAposition& pt2, EDGE*& line, AcisOptions* ao = NULL)

//连接边的实体
api_make_ewire(int num_edges, EDGE*[] edges, BODY*& body, AcisOptions* ao = NULL)

//着色
api_rh_set_entity_rgb((ENTITY*&)body,rgb_color(r,g,b));
//其中，api_gi_set_entity_rgb是着色函数，(ENTITY&)*body指定实体， body,rgb_color (r,g,b)进行着色。

//移动

SPAvector transs(i,j,k);
SPAtransf movee = translate_transf(transss);
api_transform_entity((ENTITY *&)new_body,movee);
//其中，api_copy_entity是复制函数，(ENTITY&)*pre原实体，(ENTITY *&)new_body新实体； transf movee=translate_transf(trass)是定义移方向和大小的函数；api_ transform _entity 是移位函数，把(ENTITY *&)new_body按movee规则平移。

//旋转
api_copy_entity((ENTITY *&)pre,(ENTITY *&)new_body);
transf roti=rotate_transf(arc,acis);
api_transform_entity((ENTITY *&)new_body,roti);
//其中，api_copy_entity是复制函数，(ENTITY&)pre原实体，(ENTITY *&)new_body新实体； transf roti=rotate_transf(arc,acis)是定义旋转方向和度数的函数；api_ transform _entity是旋转函数，把(ENTITY *&)new_body按roti规则旋转。

api_boolean(body_a,body_b,UNION);//求两body的并集，结果保存在body_b中
api_boolean(body_a,body_b,INTERSECTION);//求两body的交集，结果保存在body_b中
api_boolean(body_a,body_b,SUBTRACTION);//求两body的差集，结果保存在body_b中

api_sweep_with_options((ENTITY)circle, (ENTITY)arcedge, options, (BODY*&)lei);//扫略

api_find_vertex(BODY* fuwa,SPAposition(0,0,1),blv(接收));//获得实体上指定点附近的点

api_q_edges_around_vertex(v[j],&vedge_list[j]);//得到共享给定点的边

api_smooth_edge_seq(cir_edge[j],edges_to_blend[j]);//得到与给定边相连的边的序列(ENTITY_LIST&)
api_get_faces(ENTITY* ent, ENTITY_LIST& face_list, PAT_NEXT_TYPE include_pat = PAT_CAN_CREATE, AcisOptions* ao = NULL)//获取所有面

api_set_const_rounds(edges_to_blend[j],r);//设置混合半径0点y向、z向、x向，1点z向、x向，2点z向、x向，3点z向、x向

api_fix_blends(edges_to_blend[j]);//进行混合，产生曲面

api_rh_create_light(const char* type, RH_LIGHT*& light)//创建光源

api_rh_set_light_state(RH_LIGHT* light, logical on(off))//开关光源

api_rh_set_material_texture(ENTITY_LIST const& entity_list, const char* tex_name)//贴图

HA_Delete_Entity_Geometry(ENTITY* entity)//取消实体的显示

api_del_entity(ENTITY* given_entity, AcisOptions* ao = NULL)//删除实体

api_transform_entity((ENTITY *&)e,translate_transf(SPAvector(1,0,0)));//动画

api_find_vertex(e,SPAposition(0,0,5),blv);//找到实体上指定点附近的顶点，放到列表blv里

api_q_edges_around_vertex(blv,&edge_list);//寻找顶点谅连接的边。0号边是y方向(屏幕上下)，1号是z方向(屏幕内外)，2号x方向(屏幕左右)

api_smooth_edge_seq((EDGE*)edge_list[2],edges_to_blend);//得到与给定边相连的边的序列

api_set_const_rounds((ENTITY_LIST&)edges_to_blend,1.5);//设置混合半径,0点边的混合次序y向、z向、x向，注意不可以过大。

api_initialize_blending ();
api_fix_blends((ENTITY_LIST&)edges_to_blend);//进行混合，用新面取代老面3 x z y
api_terminate_blending();

//sweep扫略
sweep_options* options = ACIS_NEW sweep_options();
options->set_cut_end_off(TRUE);
api_initialize_sweeping();
  api_get_owner((ENTITY)profile, (ENTITY&)fuwa);
  api_sweep_with_options((ENTITY)profile, (ENTITY)path, options, (BODY*&)fuwa);
api_terminate_sweeping();
ACIS_DELETE options;

贴图和混合差不多。首先获取实体中的面，放到一个面的列表里。然后给这个列表的所有元素贴图。

ENTITY_LIST ball;

api_solid_block(SPAposition(0,0,0), SPAposition(5,5,5),fuwa);

api_get_faces(fuwa,ball);//获取所有面

api_rh_set_material_texture(ball,"mm.bmp");

如果需要只给几个面贴图，只需要创建另一个列表，把需要贴图的面放到里面就可以了。

Function	Description
api_boolean( )	Can be used to generate the intersection between two BODIES.
api_clash_bodies( )	Determines if two BODIES "clash", and optionally the nature of the clash.
api_clash_faces( )	Determines if two FACES "clash" and the nature of the clash.
api_edent_rel( )	Determines the spatial relationship between an EDGE and another ENTITY.
api_edfa_int( )	Generates the intersection between an EDGE and a FACE.
api_entity_entity_distance( )	Determines the minimum distance between to ENTITIES.
api_entity_entity_touch( )	Determines if two ENTITIES touch.
api_entity_point_distance( )	Determines the distance between a point and an ENTITY.
api_fafa_int( )	Generates the intersection between two faces.
api_find_cls_ptto_face( )	Determines the point on a FACE nearest a given point.
api_find_vertex( )	Determines the VERTEX on a BODY nearest a given point.
api_intersect( )	Generates the intersection between two BODIES.
api_intersect_curves( )	Determines the intersection between two EDGES or their underlying curves.
api_planar_slice( )	Generates the wire BODY representing the intersection between a plane and a BODY.
api_point_in_body( )	Determines whether a point lies inside, outside, or on the boundary of a BODY.
api_point_in_face( )	Determines whether a point lies inside, outside, or on the boundary of a FACE.The point is assumed to lie on the surface of the FACE.
api_ptent_rel( )	Determines the spatial relationship between a point and an ENTITY.