Course number: 446.633

Course title: Computer-aided design

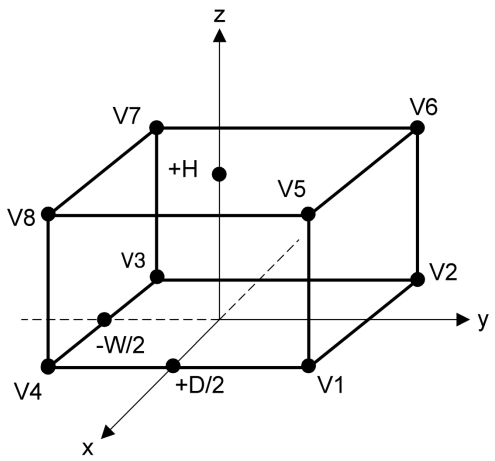
Instructor: Lee, Kunwoo

# Final Examination

Student number: _____

Name: _____

1. The following code is a procedure to create a cube. Fill in the empty blocks in the procedure with proper Euler operators together with associated arguments. Assume the vertex numbers, edge numbers, and loop numbers in the following figure in writing the arguments of the Euler operators. (10 points)




```
Body    *Create_Block (W, D, H )
        double  W, D, H;
        {
                Body    *B;
                Shell   *S;
                Loop    *L1, *L2, *L3, *L4, *L5, *L6;
                Edge    *E1, *E2, *E3, *E4, *E5, *E6, *E7, *E8, *E9, *E10, *E11,*E12;
                Vertex  *V1, *V2, *V3, *V4, *V5, *V6, *V7, *V8;

                B = malloc ( sizeof ( Body ) );
                MEVVLS ( B, &E1, &V1, &V2, &L1, &S, D/2, W/2, 0, -D/2, W/2, 0 );
                (a)  (4 points)
                MEV ( B, L1, V3, &E3, &V4, D/2, -W/2, 0 );
                MEL ( B, L1, V4, V1, &E4, &L2 );
                (b)  (3 points)
                MEV ( B, L1, V2, &E6, &V6, -D/2, W/2, H );
                MEV ( B, L1, V3, &E7, &V7, -D/2, -W/2, H );
```

```
        MEV ( B, L1, V4, &E8, &V8, D/2, -W/2, H );
        (c) (3 points)
        MEL ( B, L1, V6, V7, &E10, &L4 );
        MEL ( B, L1, V7, V8, &E11, &L5 );
        MEL ( B, L1, V8, V5, &E12, &L6 );
        return ( B );
}
```

2. Answer the following questions for a non-uniform non-periodic B-spline curve of order 4 defined by the control points $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$, $\mathbf{P}_4$, and $\mathbf{P}_5$. (25 points)

   (a) What are the knot values? (5 points)
   (b) How many different curves is this composite curve made of? (5 points)
   (c) Sketch the blending function $N_{3,1}$, $N_{4,1}$, and $N_{3,2}$. Use the following equation. (15 points)

$$N_{i,k}(u) = \frac{(u - t_i)N_{i,k-1}(u)}{t_{i+k-1} - t_i} + \frac{(t_{i+k} - u)N_{i+1,k-1}(u)}{t_{i+k} - t_{i+1}}$$

$$N_{i,1}(u) = \begin{cases} 1 & t_i \le u \le t_{i+1} \\ 0 & otherwise \end{cases}$$

3. Prove the following formula for the differentiation of the Bezier curve. (15 points)

$$\frac{d\mathbf{P}(u)}{du} = n \sum_{i=0}^{n-1} \binom{n-1}{i} u^i (1-u)^{n-1-i} \mathbf{a}_i$$

$$where \quad \mathbf{a}_i = \mathbf{P}_{i+1} - \mathbf{P}_i \quad i = 0, 1, \dots, n-1$$

Bezier curve equation is given as below.

$$\mathbf{P}(u) = \sum_{i=0}^{n} \binom{n}{i} u^i (1-u)^{n-i} \mathbf{P}_i \quad (0 \le u \le 1)$$

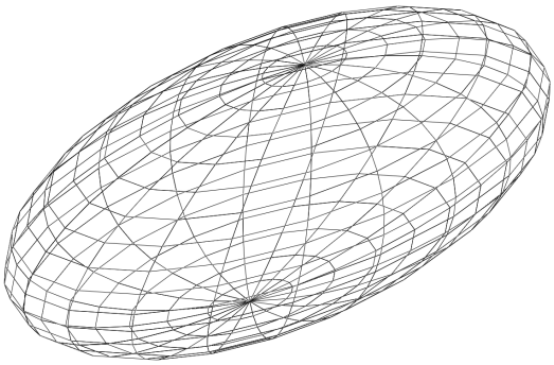
↑ Control point

4. Derive the transformation matrix $T_{w\text{-}v}$ that transforms the coordinates in world coordinates to those in viewing coordinates when the locations of the view point and the view site are specified to be (10, 0, 0) and (0, 0, 0) respectively with respect to the world coordinate system. Assume that z axis of the world coordinate system is desired to be the up vector. (20 points)

5. (OpenGL) In OpenGL, there is no default function which draws an ellipsoid. So, let's define our own function which draws an ellipsoid using OpenGL primitives as below. (10 points)

```
void DrawHalfEllipsoid(double a, double b, double c);
void DrawFullEllipsoid(double a, double b, double c);
```

Use the following hint.



Equation of ellipsoid is: $\left(\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}=1\right)$

Ellipsoid can be parameterized by two parameters: $(\theta, \emptyset)$

For instance, $z = c\sin\emptyset\ (-\frac{\pi}{2} \le \emptyset \le \frac{\pi}{2})$ .

From this result, the other variables can be parameterized.

Following is the skeleton code of the functions to be implemented.

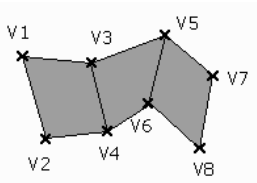
```
void DrawHalfEllipsoid(double a, double b, double c)
{
        // Define PI value: PI=3.1415926535....
        const double PI_VAL = 4.0 * atan(1.0);
        // Variable for ∅
        double Phi = 0;
        // Variable for Θ
        double Theta = 0;
        int SegmentsNumber_Vertical = 10;
        int SegmentsNumber_Horizontal = 20;
        double step_Phi = 0.5 * PI_VAL / (double)SegmentsNumber_Vertical;
        double step_Theta = 2.0 * PI_VAL / (double)SegmentsNumber_Horizontal;

        glBegin(Mode_1);
        for(Phi = 0; Phi < 0.5 * PI_VAL ; Phi+=step_Phi)
        {
                for(Theta = 0; Theta <= 2.0 * PI_VAL ; Theta+=step_Theta)
                {
                        double low_x = Expression_1;
                        double low_y = Expression_2;
                        double low_z = Expression_3;
                        double high_x = Expression_4;
                        double high_y = Expression_5;
                        double high_z = Expression_6;
                        glColor3d(0.0, 0.0, 0.0);
                        glVertex3d(low_x, low_y, low_z);
                        glVertex3d(high_x, high_y, high_z);
                }
```

```
        }
        glEnd();
}

void DrawFullEllipsoid(double a, double b, double c)
{
        glPushMatrix();
                DrawHalfEllipsoid(a,b,c);
                glPushMatrix();
                glRotated(Expression_7);
                DrawHalfEllipsoid(a,b,c);
                glPopMatrix();
        glPopMatrix();
}
```

(a) Write the proper *Mode_1* for glBegin() function calls. Choose the OpenGL geometric primitive type which shows the pattern like this. (1 point)




(b) Write the proper *Expression_1 ~ Expression_6* in terms of given function input parameters, local function variables and trigonometric functions, sin( ) and cos( ). Note that trigonometric functions in *c++* use radians for their angle units. (8 points)

(c) Write the proper *Expression_7* in terms of proper numerals. Note that OpenGL functions use degrees for their angle units. (1 point)

6. (SolidWorks API) Answer the following questions regarding the SolidWorks API code given below. (10 point)

```
Option Explicit

Dim swApp As SldWorks.SldWorks
Dim swModel As SldWorks.ModelDoc2
Dim swFeatMgr As SldWorks.FeatureManager

Sub main()
    Set swApp = Application.SldWorks
    Set swModel = swApp.NewPart
    Set swFeatMgr = swModel.FeatureManager

    swModel.CreateCircle2 0.04, 0.03, 0, 0.048, 0.036, 0
    swModel.SketchOffset2 0.002, 0, 1
    swModel.CreateLine2(0, 0, 0, 0, 0.05, 0).ConstructionGeometry = True
    swFeatMgr.FeatureRevolve 90 * 3.14 / 180, True, 0, 0, 0, True, False, True
End Sub
```

(a) There is another way to create a circle using an API call which is more easily applicable to user interface. Replace the line, 'swModel.CreateCircle2 0.04, 0.03, 0, 0.048, 0.036, 0', by using a 'CreateCircleByRadius2' API call. The changed line should create a same circle with the previous one. (3 points)

(b) Draw the result of the code in a wire-frame view. In the sketch, you should indicate coordinate system with axes and every dimension which are related with the location of the center of circle, the length of radius and offset and the angle of revolution. (7 points)

7. (Parasolid API) Answer the following questions regarding the Parasolid API code given below. (10 point)

```
int finished=0;
static PK_BODY_t block, cylinder, plane;
PK_PARTITION_t partition;
static PK_PMARK_t pmark;
PK_TOPOL_track_r_t tracking;
static PK_boolean_r_t results1;
PK_section_2_r_t results2;
PK_boolean_r_t results3;
PK_AXIS2_sf_t basis_set1, basis_set2;
PK_BODY_boolean_o_t booleanOpt;
int n_new_entities=0, n_mod_entities=0, n_del_entities=0;
PK_ENTITY_t *new_entities=NULL, *mod_entities=NULL, *del_entities=NULL;

switch( step )
{
case 1:
    PK_BODY_create_solid_block(20.0, 20.0, 5.0, NULL, &block);

    basis_set1.location.coord[0] = -5;
    basis_set1.location.coord[1] = 5;
    basis_set1.location.coord[2] = 0;
    basis_set1.axis.coord[0] = 0;
    basis_set1.axis.coord[1] = 0;
    basis_set1.axis.coord[2] = 1;
    basis_set1.ref_direction.coord[0] = 1;
    basis_set1.ref_direction.coord[1] = 0;
    basis_set1.ref_direction.coord[2] = 0;
    PK_BODY_create_solid_cyl(4, 15, &basis_set1, &cylinder);

    PK_SESSION_ask_curr_partition(&partition);
    PK_PARTITION_make_pmark(partition, &pmark);

    PK_BODY_boolean_o_m(booleanOpt);
    booleanOpt.merge_imprinted = PK_LOGICAL_true;
    PK_BODY_boolean_2(block, 1, &cylinder, &booleanOpt, &tracking, &results1);
```

```
    break;

case 2:
    basis_set2.location.coord[0] = 0;
    basis_set2.location.coord[1] = 0;
    basis_set2.location.coord[2] = 0;
    basis_set2.axis.coord[0] = 1;
    basis_set2.axis.coord[1] = 0;
    basis_set2.axis.coord[2] = 0;
    basis_set2.ref_direction.coord[0] = 0;
    basis_set2.ref_direction.coord[1] = 0;
    basis_set2.ref_direction.coord[2] = 1;
    PK_BODY_create_sheet_rectangle(30, 30, &basis_set2, &plane);

    PK_BODY_section_o_t sectionOpt;
    PK_BODY_section_o_m(sectionOpt);
    sectionOpt.fence = PK_section_fence_back_c;
    PK_BODY_section_with_sheet_2(results1.bodies[0], plane, &sectionOpt, &tracking,
&results2);
    break;

case 3:
    PK_PMARK_goto(pmark, &n_new_entities, &new_entities, &n_mod_entities,
&mod_entities, &n_del_entities, &del_entities);
    if(n_new_entities) PK_MEMORY_free(new_entities);
    if(n_mod_entities) PK_MEMORY_free(mod_entities);
    if(n_del_entities) PK_MEMORY_free(del_entities);

    PK_BODY_boolean_o_m(booleanOpt);
    booleanOpt.function = PK_boolean_subtract_c;
    PK_BODY_boolean_2(block, 1, &cylinder, &booleanOpt, &tracking, &results3);
    break;

default:
    finished = 1;
}

return finished;
```

Draw the result of the code in a wire-frame view. In the sketch, you should indicate coordinate system with axes. Do NOT include dimensions.

(a) Draw the result after 'case 1' is finished. (3 points)

(b) Draw the result after 'case 2' is finished. (3 points)

(c) Explain the role of function 'PK_PMARK_goto' briefly and draw the result after 'case 3' is finished. (4 points)

■ **SolidWorks API Help**

ModelDoc2::CreateCircle2

Description:
This method creates a circle based on a center point and a point on the circle.
Syntax:
retval = ModelDoc2.CreateCircle2 ( xc, yc, zc, xp, yp, zp )
Input:
(double) xc – X value of the circle center point, in meters
(double) yc – Y value of the circle center point, in meters
(double) zc – Z value of the circle center point, in meters
(double) xp – X value of the point on the circle, in meters
(double) yp – Y value of the point on the circle, in meters
(double) zp – Z value of the point on the circle, in meters
Return:
(LPDISPATCH) retval – Pointer to the Dispatch object of the circle that was created

ModelDoc2::CreateCircleByRadius2

Description:
This method creates a circle based on a center point and a specified radius.
Syntax:
retval = ModelDoc2.CreateCircleByRadius2 ( xc, yc, zc, radius )
Input:
(double) xc – X value of the circle center point in meters
(double) yc – Y value of the circle center point in meters
(double) zc – Z value of the circle center point in meters
(double) radius – radius of the circle in meters
Return:
(LPDISPATCH) retval – Pointer to the Dispatch object of the circle that was created

ModelDoc2::SketchOffset2

Description:
This method offsets the selected sketch segments.
Syntax:
retval = ModelDoc2.SketchOffset2 ( offset, bothDirections, chain )
Input:
(double) offset – Desired offset value; negative value offsets in opposite direction
(BOOL) bothDirections – TRUE to offset in both directions, FALSE to not
(BOOL) chain – TRUE if you want entire chain of entities offset, FALSE if you want only selected sketch entities offset
Return:
(BOOL) retval – TRUE if the offset is successful, FALSE if not

ModelDoc2::CreateLine2

Description:
This method creates a sketch line in the currently active 2D or 3D sketch.
Syntax:
retval = ModelDoc2.CreateLine2 ( xStart, yStart, zStart, xEnd, yEnd, zEnd )
Input:
(double )p1x – X value of the line start point
(double) p1y – Y value of the line start point
(double) p1z – Z value of the line start point
(double) p2x – X value of the line end point
(double) p2y – Y value of the line end point
(double) p2z – Z value of the line end point
Return:
(LPDISPATCH) retval – Pointer to a Dispatch object, the newly created SketchSegment object; if the operation fails, then NULL is returned

FeatureManager::FeatureRevolve

Description:
This method creates a revolved feature, which can be a base feature or a boss feature.
Syntax:
retVal = FeatureManager.FeatureRevolve ( angle, reverseDir, angle2, revType, options, merge, useFeatScope, useAutoSel )
Input:
(double) angle – Angle of revolution in radians
(VARIANT_BOOL) reverseDir – TRUE reverses the angle direction, FALSE does not
(double) angle2 – Angle of revolution in radians
(long) revType – Type of revolution (see Remarks)
(long) options – Additional control
(VARIANT_BOOL) merge – TRUE to merge the results in a multibody part, FALSE to not
(VARIANT_BOOL) useFeatScope – TRUE if the feature only affects selected bodies, FALSE if the feature affects all bodies
(VARIANT_BOOL) useAutoSel – TRUE to automatically select all bodies and have the feature affect those bodies, FALSE to select the bodies the feature affects (see Remarks)
Output:
(LPFEATURE) *retVal –Pointer to the Feature object
Remarks:
The revType argument can be one of the following values:
0 = One direction revolution
1 = Mid-plane revolution. For this type of revolve, the angle specification specifies the full revolution. The angle to be revolved is (angle/2) on either side of the sketch. The reverseDir argument has no affect.
2 = Two direction revolution. For a two direction revolve, the angle is the angle to be revolved in direction 1 and angle2 is the angle to be revolved in direction 2.

■ **Parasolid API Help**

PK_BODY_create_solid_block

This function creates a solid block.
--- received arguments ---
double x, --- block extent in local x direction (>0)
double y, --- block extent in local y direction (>0)
double z, --- block extent in local z direction (>0)
const PK_AXIS2_sf_t *basis_set, --- position and orientation (may be NULL)
--- returned arguments ---
PK_BODY_t *const body --- solid body returned

In its local coordinate system, this function creates a solid block whose base is centred at the origin, and whose total length in the directions of the local axes is 'x', 'y' and 'z'.

'basis_set' positions and orientates the local coordinate system in the world coordinate system. If it is given as NULL, then the local and world coordinate systems are the same.

PK_BODY_create_solid_cyl

This function creates a solid cylinder.
--- received arguments ---
double radius, --- cylinder radius (>0)
double height, --- cylinder height (>0)
const PK_AXIS2_sf_t *basis_set --- position and orientation (may be NULL)
--- returned arguments ---
PK_BODY_t *const body --- solid body returned

In its local coordinate system, this function creates a solid cylinder whose base of the given 'radius' is centred at the origin and lies in the xy plane, and which extends to the given 'height' along the z axis.

'basis_set' positions and orientates the local coordinate system in the world coordinate system. If it is given as NULL, then the local and world coordinate systems are the same.

The basis_set in the standard forms of the created CYL surface and CIRCLE curves inherits the ref_direction from the given 'basis_set'.

PK_BODY_boolean_2

This function performs a boolean operation between the target body and the list of tool bodies.
--- received arguments ---
PK_BODY_t target, --- body to receive message
int n_tools, --- number of tool bodies
const PK_BODY_t tools[], --- tool bodies
const PK_BODY_boolean_o_t *options, --- boolean options

--- returned arguments ---
PK_TOPOL_track_r_t *const tracking, --- tracking information
PK_boolean_r_t *const results --- boolean results

struct PK_BODY_boolean_o_s

'merge_imprinted' --- Merge all mergeable imprinted edges created and/or located by the boolean operation. Note that an existing edge can be considered an imprinted edge if it forms part of the intersection between faces from the target and tool.

'function' --- Indicates whether an intersect, unite or subtract operation is to be performed, permitted values are PK_boolean_intersect_c, PK_boolean_subtract_c, PK_boolean_unite_c

PK_BODY_create_sheet_rectangle

This function creates a sheet rectangle.
--- received arguments ---
double x, --- x dimension of rectangle (>0)
double y, --- y dimension of rectangle (>0)
const PK_AXIS2_sf_t *basis_set, --- position and orientation (may be NULL)
--- returned arguments ---
PK_BODY_t *const body --- sheet body returned

In its local coordinate system, this function creates a sheet rectangle in the xy plane centred at the origin, and whose total lengths in the local x and y directions are 'x' and 'y'.

'basis_set' positions and orientates the local coordinate system in the world coordinate system. If it is given as NULL, then the local and world coordinate systems are the same.

PK_BODY_section_with_sheet_2

This function sections the target body with the tool sheet body.
--- received arguments ---
PK_BODY_t target, --- body to be sectioned by sheet
PK_BODY_t sheet, --- sectioning sheet body
const PK_BODY_section_o_t *options, --- sectioning options
--- returned arguments ---
PK_TOPOL_track_r_t *const tracking, --- tracking information
PK_section_2_r_t *const results --- front/back faces/bodies

struct PK_BODY_section_o_s

'fence' --- To determine which resultant bodies or faces are returned when performing a body sectioning operation. Permitted values are: PK_section_fence_front_c(return bodies in front), PK_section_fence_back_c(return bodies behind), and PK_section_fence_both_c(return all bodies)