CompSci 372 S1 C 2005
Computer Graphics

Mid Term Test – Friday, 29th April 2005, Sample Solution

Surname (Family Name):	
First Name(s):	
ID Number:	
Login Name (UPI):	

Instructions:

1. Attempt **ALL** questions.
2. The test is for one (1) hour.
3. This is a closed book test.
4. Calculators are **NOT** permitted.
5. Write your answers in the spaces provided. There is space at the back for answers that overflow the allotted space.
6. **Questions total 50 Marks**.
7. This test is worth 10% of your final marks for CompSci372 S1 C

Section	Marks	Maximum Marks
Q.1		**8**
Q.2		**9**
Q.3		**9**
Q.4		**14**
Q.5		**10**
Total		**50**

Question 1 – Short answer test [8 marks]

Complete each of the following statements by filling in the underlined blank spaces. Each blank space is worth 1 mark.

[8 marks]

(a) The **frame buffer** is a region of memory sufficiently large to hold all pixels of the display. The OpenGL command `glClear(GL_COLOR_BUFFER_BIT)` clears a region in it which corresponds to the current display window.

(b) Given are $\mathbf{u} = \begin{pmatrix} 0 \\ -2 \\ 1 \end{pmatrix}$ and $\mathbf{v} = \begin{pmatrix} 0 \\ 3 \\ 2 \end{pmatrix}$.

(i) The dot product of the two vectors is **u.v** = **0*0+(-2)*3+1*2 = -4** .

(ii) The vector product of the two vectors is $\mathbf{u}\times\mathbf{v} = \mathbf{v} = \begin{pmatrix} (-2)*2-1*3 \\ 1*0-2*0 \\ 0*3-(-2)*0 \end{pmatrix} = \begin{pmatrix} -7 \\ 0 \\ 0 \end{pmatrix}$.

(c) Given are two 3D orthogonal vectors **a** and **b** which both have a length of 3 units.

Then $|\mathbf{a}\times\mathbf{b}|$ = **9** (your answer must be a number and not a formula).

(d) Given two spheres with the centres $\mathbf{c}_1$ and $\mathbf{c}_2$ and the radii r_1 and r_2. Then the distance d between the two spheres is $d = |\mathbf{c}_1 - \mathbf{c}_2| - r_1 - r_2$.

(e) Assumed an object is transformed by the matrix **M**. Then the surface normals of this object must be transformed by the matrix $\left(\mathbf{M}^{-1}\right)^T$.

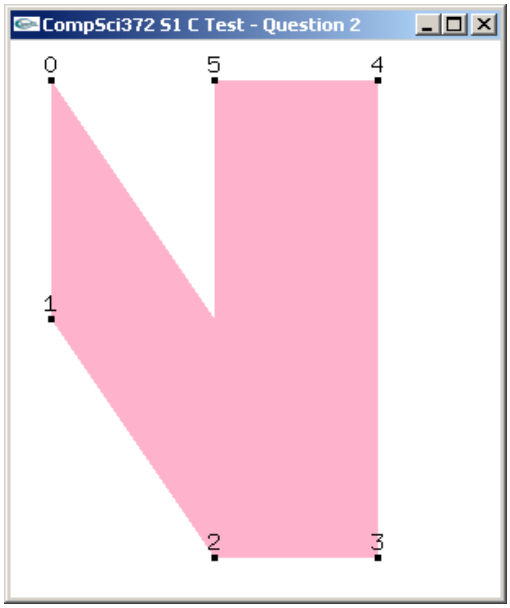
(f) Specify a 3×3 rotation matrix R which rotates the *x*-axis into the *y*-axis.

$\mathbf{R} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ (the solution is just a 90 degree rotation around the z-axis. We can obtain the solution easily from the 2D rotation matrix $\begin{pmatrix} \cos 90^0 & -\sin 90^0 \\ \sin 90^0 & \cos 90^0 \end{pmatrix}$)

(g) Describe in one sentence what the command glPushMatrix() is doing. Answer: **It makes a copy of the currently active matrix stack and pushes it on top of the stack**.

Question 2 – OpenGL [9 marks]

In this question you must write code that draws the 2D object shown on the right. The vertices are stored in a global 2D float array **v** in the order indicated in the figure on the right. For each answer use the most efficient representation.



If you want to save yourself some writing you may assume that the following function is defined:

```
void p(int i){
 glVertex2fv(v[i]);
}
```

A. Complete the display function shown below so that it draws the above shape using the **GL_QUADS** mode and the **glVertex2fv** command (or the function p) [3 marks].

```
void display(void)
{
    // clear all pixels in frame buffer
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.1, 0.2);    // reddish colour
    glBegin(GL_QUADS);
```

```
    glVertex2fv(v[1]);    // or p(1);
    glVertex2fv(v[2]);    // or p(2);
    glVertex2fv(v[3]);    // etc.
    glVertex2fv(v[0]);
    glVertex2fv(v[2]);
    glVertex2fv(v[3]);
    glVertex2fv(v[4]);
    glVertex2fv(v[5]);
```

```
    glEnd();
    glFlush();
}
```

B. Complete the display function shown below so that it draws the above shape using the **GL_QUAD_STRIP** mode and the **glVertex2fv** command (or the function p) [3 marks].

```
void display(void)
{
    // clear all pixels in frame buffer
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.1, 0.2);  // reddish colour
    glBegin(GL_QUAD_STRIP);
```

```
    glVertex2fv(v[1]);
    glVertex2fv(v[0]);
    glVertex2fv(v[2]);
    glVertex2fv(v[3]);
    glVertex2fv(v[5]);
    glVertex2fv(v[4]);
```

```
    glEnd();
    glFlush();
}
```

C. Complete the display function shown below so that it draws the above shape using the **GL_TRIANGLE_STRIP** mode and the **glVertex2fv** command (or the function p) [3 marks].

```
void display(void)
{
    // clear all pixels in frame buffer
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.1, 0.2);  // reddish colour
    glBegin(GL_TRIANGLE_STRIP);
```

```
    glVertex2fv(v[1]);
    glVertex2fv(v[0]);
    glVertex2fv(v[2]);
    glVertex2fv(v[3]);
    glVertex2fv(v[5]);
    glVertex2fv(v[4]);
```

```
    glEnd();
    glFlush();
}
```

Question 3 – 3D Geometry [9 marks]

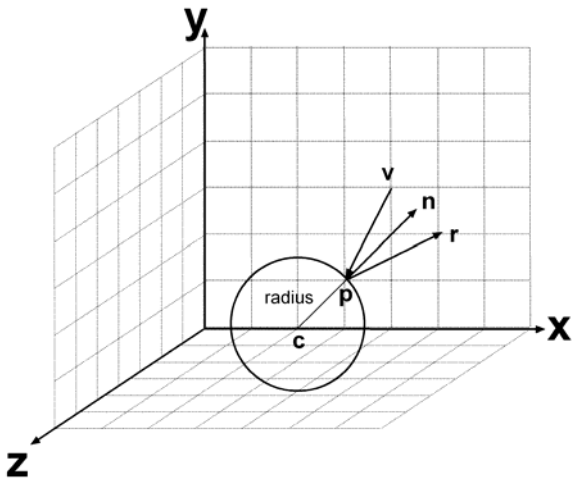
A. Given is a line $\mathbf{p}(t) = \begin{pmatrix}1\\2\\0\end{pmatrix} + t\begin{pmatrix}1\\0\\2\end{pmatrix}$ which is parallel to the plane $\begin{pmatrix}2\\2\\-1\end{pmatrix}\bullet\mathbf{p}=2$. Compute the distance between the line and the plane.

[3 marks]

Since the line is parallel to the plane the distance between the two objects is given by the distance of any point on the line to the plane, e.g.

$$\text{distance} = \frac{\mathbf{n}\bullet\mathbf{p}(0) - d}{|\mathbf{n}|} = \frac{\begin{pmatrix}2\\2\\-1\end{pmatrix}\bullet\begin{pmatrix}1\\2\\0\end{pmatrix} - 2}{\left|\begin{pmatrix}2\\2\\-1\end{pmatrix}\right|} = \frac{6-2}{3} = \frac{4}{3}$$

B. Given is a sphere with the centre $\mathbf{c} = \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}$ and the radius $r = \sqrt{2}$. A spherical object flying with the direction $\mathbf{v} = \begin{pmatrix} -1 \\ -2 \\ 0 \end{pmatrix}$ hits the sphere at the point $\mathbf{p} = \begin{pmatrix} 3 \\ 1 \\ 0 \end{pmatrix}$ and is reflected on it. Compute the direction of the object after reflection. Clearly explain the different steps of your computation. [6 marks]



In order to compute the direction after reflection we have to compute first the normal *n* of the sphere at the hit point:

$$\mathbf{n} = \mathbf{p} - \mathbf{c} = \begin{pmatrix} 3 \\ 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \text{ and after normalisation } \hat{\mathbf{n}} = \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \\ 0 \end{pmatrix}$$

Using the reflection formula we get:

$$\mathbf{r} = \mathbf{v} - 2(\mathbf{v} \bullet \hat{\mathbf{n}})\hat{\mathbf{n}} = \begin{pmatrix} -1 \\ -2 \\ 0 \end{pmatrix} - 2\left(\begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \\ 0 \end{pmatrix} \bullet \begin{pmatrix} -1 \\ -2 \\ 0 \end{pmatrix}\right)\begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \\ 0 \end{pmatrix} = \begin{pmatrix} -1 \\ -2 \\ 0 \end{pmatrix} + 6/\sqrt{2} \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix}$$

Question 4 – Transformations [14 marks]

A. Given is a function

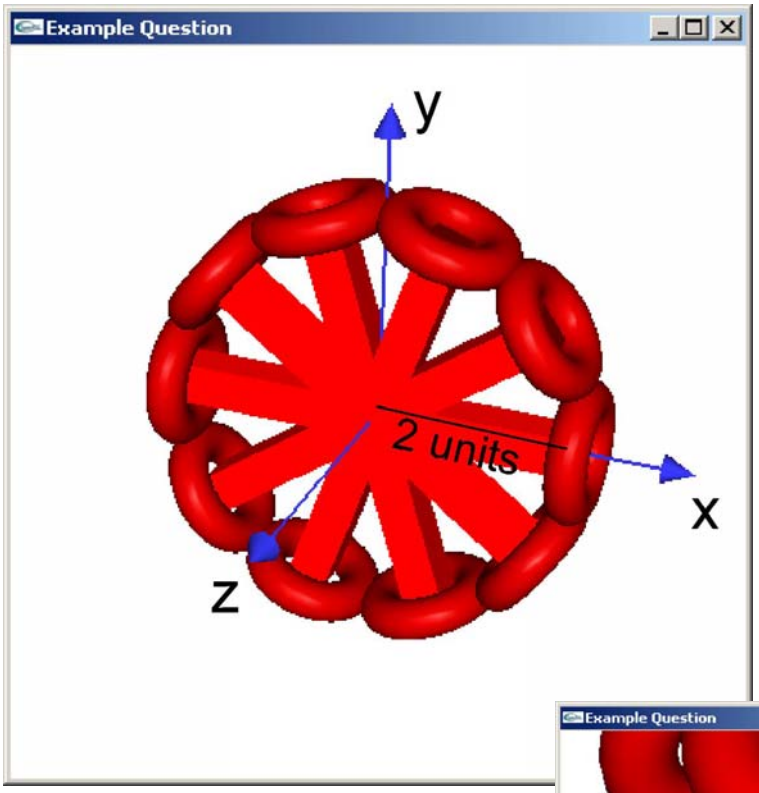
```
drawCube()
```

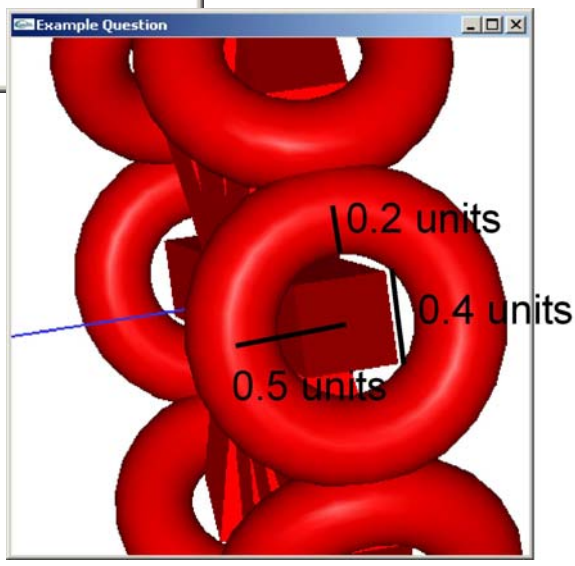
which draws an axis-aligned **unit cube** with side length 1 **centred** at the origin, and the function

```
drawTorus(float innerRadius, float outerRadius)
```

which draws a torus with the given inner and outer radius and which is centred at the origin and has a centre axis aligned with the **z-axis**.

Use these functions to complete the display method on the next page so that it draws the object shown in the images below. The object consists of cuboids with a length of 2 units and a base of 0.4×0.4 units and torii with an inner radius of 0.2 units and an outer radius of 0.5 units. The whole object consists of *n* cubes and torii which are evenly distributed in a circular fashion around the z-axis as shown in the images below. The centre of the bottom surfaces of the cuboids is the origin [7 marks].





```
#include <math.h>

void display(void)
{
    glMatrixMode( GL_MODELVIEW ); // Set the view matrix ...
    glLoadIdentity();                    // ... to identity.
    gluLookAt(0,0,20, 0,1.25,0, 0,1,0);  // camera is on the z-axis
    trackball.tbMatrix();                // rotate the scene using the trackball ...
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_NORMALIZE);

    // Draw the scene
```

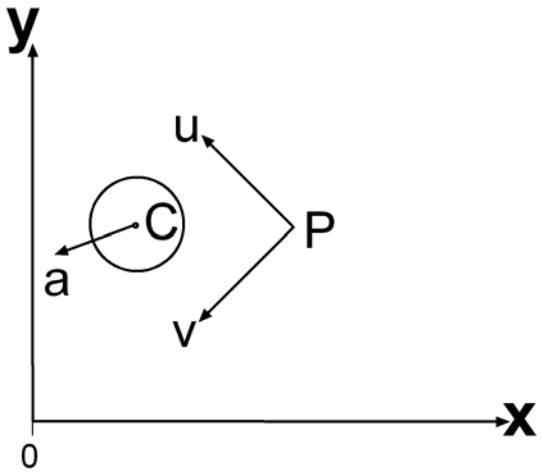
```
    int n=10;       // your code should work for any value n≥1

    for(int i=0;i<n;i++)
    {
      glPushMatrix();
      glRotatef(360.0*i/n,0,0,1);
      glPushMatrix();
      glScalef(2, 0.4, 0.4);
      glTranslatef(0.5,0,0);
      drawCube();
      glPopMatrix();
      glTranslatef(2,0,0);
      glRotatef(90,0,1,0);
      drawTorus(0.2, 0.5);
      glPopMatrix();
    }
```

```
    glDisable(GL_NORMALIZE);
    glFlush ();
    glutSwapBuffers();
}
```

B. Given is local coordinate system with the origin $P=(p_x,p_y)^T$ and the basis vectors $\mathbf{u}=(u_x, u_y)^T$ and $\mathbf{v}=(v_x, v_y)^T$, which are orthogonal and have a length of one.

An animated object is defined with the center $C=(c_u,c_v)^T$ and the direction $\mathbf{a}=(a_u, a_v)^T$ which are both given in uv-coordinates.



(i) Write down the homogeneous 2D transformation matrix **M**, which transforms the centre point C from uv-coordinates into xy-coordinates. You are allowed to write the transformation matrix as a product of simpler matrices (i.e. you are not required to multiply the matrices)

[4 marks]

This case is just the opposite from the "wall coordinate transformation" in assignment 2. We have the uv-coordinates and we want to determine the world coordinates. This is equivalent with transforming the xy-coordinate system into the uv-coordinate system which is achieved by first rotating it and then translating it by p. Hence

$$\mathbf{M} = \mathbf{T}_P\mathbf{R}_{xy\_to\_uv} = \begin{pmatrix} 1 & 0 & p_x \\ 0 & 1 & p_y \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} u_x & v_x & 0 \\ u_y & v_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(ii) Write down the homogeneous 2D transformation matrix **N**, which transforms the direction vector **a** from uv-coordinates into xy-coordinates. You are allowed to write the transformation matrix as a product of simpler matrices (i.e. you are not required to multiply the matrices).

[3 marks]

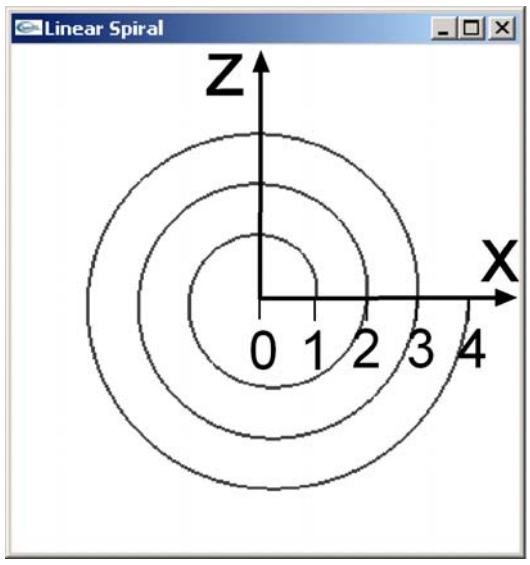
A direction vector is independent of its position; hence we only have to apply the rotation shown above.

$$\mathbf{N} = \mathbf{R}_{xy\_to\_uv} = \begin{pmatrix} u_x & v_x & 0 \\ u_y & v_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Question 5 – Modelling (Curves and Surfaces) [10 marks]

A. Determine the parametric equation $\mathbf{c}(t)=(x(t),y(t),z(t))$ of the spiral curve shown in the image on the right. The spiral has 3 revolutions, it starts at the point $(1,0,0)^T$ and its radius increases **linearly** from 1 to 4. The spiral lies in the xz-plane and it is centred at the origin.

[3 marks]



The parametric equation of the spiral is obtained by starting with the equation of a circle (with 3 revolutions) and then using a radius a function which increases linearly from 1 to 4. Hence we get:

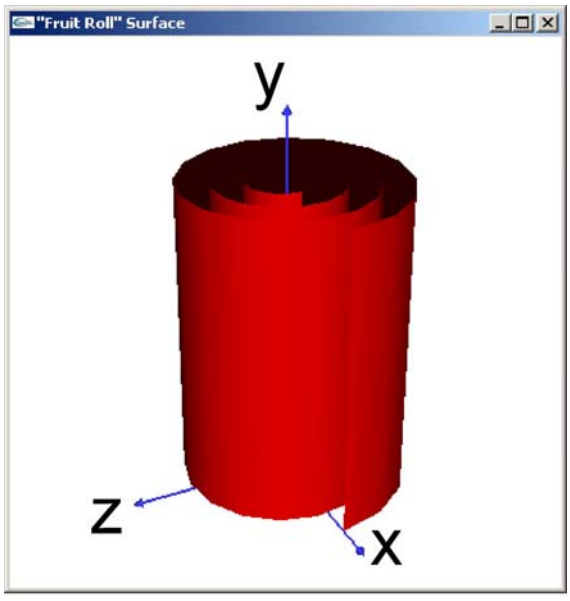
$$\mathbf{c}(t)=\begin{pmatrix}(t+1)\cos(2\pi t)\\0\\(t+1)\sin(2\pi t)\end{pmatrix},\quad t\in[0,3]$$

B. Given is the function

```
float* curve(float t)
```

which implements the curve $\mathbf{c}(t)$ $(0\le t\le 1)$ from part (A) of this question. The function returns an array of three float values.

Complete the code below which draws the "fruit roll" like surface shown in the image on the right. The "fruit roll" has a height of 10 units and its cross section is the curve from part (A).
Note that you are allowed to approximate the normal at a point **p** by the normal of the cylinder having the same distance as **p** to the y-axis. [7 marks]



```
const float Pi = 3.14159265358979323846264338327f;

void display(void)
{
   glMatrixMode( GL_MODELVIEW );  // Set the view matrix ...
   glLoadIdentity();              // ... to identity.
   gluLookAt(0,10,40, 0,5,0, 0,1,0); // camera is on the z-axis
   trackball.tbMatrix();          // rotate the cylinder using the trackball ...

   glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, mat_ambient_and_diffuse);

   // Draw the "fruit roll"
   float height=10.0;
```

```
int nSteps=50;
float *c;
float t,length;
glBegin(GL_QUAD_STRIP);
for(int i=0;i<nSteps;i++)
{
   t=(float) i/(nSteps-1);
   c=curve(t);
   length=sqrt(c[0]*c[0]+c[1]*c[1]+c[2]*c[2]);
   glNormal3f(c[0]/length, 0, c[2]/length);
   glVertex3f(c[0],height,c[2]);
   glVertex3f(c[0],0,c[2]);
}
glEnd();
```

```
   glFlush ();
   glutSwapBuffers();
   }
```