CompSci 372 S1 C 2004
Computer Graphics

Mid Term Test 30th April 2004, 6.30 pm – 7.30 pm

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		10
Q.2		10
Q.3		10
Q.4		10
Q.5		10
Total		50

Question 1 – Short answer test [10 marks]

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

[10 marks]

(a) Given are $\mathbf{u} = \begin{pmatrix} 3 \\ 0 \\ -4 \end{pmatrix}$ and $\mathbf{v} = \begin{pmatrix} -2 \\ 0 \\ 8 \end{pmatrix}$.

(i) The dot product of the two vectors is **u.v** = **3*(-2)+(-4)*8=-38** .

(ii) The vector product of the two vectors is $\mathbf{u} \times \mathbf{v} = \begin{pmatrix} 0*8-(-4)*0 \\ (-4)*(-2)-8*3 \\ 3*0-(-2)*0 \end{pmatrix} = \begin{pmatrix} 0 \\ -16 \\ 0 \end{pmatrix}$.

(iii) The so-called outer product is computed as $\mathbf{u}\,\mathbf{v}^T = \begin{pmatrix} 3 \\ 0 \\ -4 \end{pmatrix} \begin{pmatrix} -2 & 0 & 8 \end{pmatrix} = \begin{pmatrix} -6 & 0 & 24 \\ 0 & 0 & 0 \\ 8 & 0 & -32 \end{pmatrix}$.

(b) Given are two 3D vectors **a** and **b**.

(i) Then (**a** × **b**) .**a**= **0** .

(ii) Assumed **a.b**=0 and **a** × **b**=**0** (the zero vector). What statement can you make in this case about **a** and **b**? Answer: **At least one of these vectors must be 0** .

(c) The normal at each point of a parametric surface **p**(s,t) is given by $\mathbf{n}(s,t) = \left(\frac{\partial \mathbf{p}}{\partial t}\right) \times \left(\frac{\partial \mathbf{p}}{\partial s}\right)$.

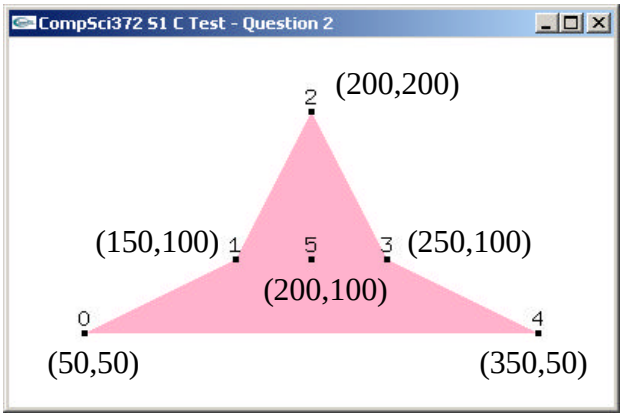
(d) Given is a sphere with the centre **c** and a point **p** on its surface. Then the surface normal **n** at the point **p** is **n**= **p**- **c**.

(e) The parametric equation for a quarter circle from point $\mathbf{p}_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ to $\mathbf{p}_1 = \begin{pmatrix} -1 \\ 2 \end{pmatrix}$ with centre $\mathbf{c} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$ is $\mathbf{p}(t) = \begin{pmatrix} 2\cos j + 1 \\ 2\sin j + 2 \end{pmatrix},\ j \in \left[p, \frac{3}{2}p\right]$ [2 marks].

(f) Describe in one sentence what the command glPopMatrix() is doing. Answer: **Removes the top element of the current matrix stack** .

Question 2 – OpenGL [10 marks]

In this question you have to draw the 2D object shown on the right. The vertices 0 to 5 of the object and the corresponding co-ordinates are shown in the image on the right. For each answer to the questions below use the most efficient representation.



A. Complete the code fragment below so that it defines a global array containing the six 2D vertices of the above shape in the given order [2 marks]:

```
const int numVertices=6;
const float v[numVertices][2] = {{50, 50}, {150, 100}, {200, 200}, {250, 100},
                                 {350, 50}, {200, 100}};
```

B. Complete the display function below so that it draws the given shape using the **GL_TRIANGLE_FAN** mode and the **glVertex2fv** command. The vertex numbers and the black dots in the above image have been inserted afterwards for clarity and you don't have to draw them. [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_FAN);
    glVertex2fv(v[1]);
    glVertex2fv(v[0]);
    glVertex2fv(v[4]);
    glVertex2fv(v[3]);
    glVertex2fv(v[2]);
    glEnd();
    glFlush();
}
```

C. Complete the display function below so that it draws the given shape using the **GL_TRIANGLE_STRIP** mode and the **glVertex2fv** command [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[0]);
glVertex2fv(v[4]);
glVertex2fv(v[1]);
glVertex2fv(v[3]);
glVertex2fv(v[2]);
```

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

D. Complete the display function below so that it draws the given shape using the **GL_QUAD_STRIP** mode and the **glVertex2fv** command [2 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[0]);
glVertex2fv(v[4]);
glVertex2fv(v[1]);
glVertex2fv(v[3]);
glVertex2fv(v[2]);
glVertex2fv(v[2]);
```

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

Question 3 – 3D Geometry [10 marks]

A. Find the equation $\mathbf{n} \bullet \mathbf{p} = d$ of the plane which goes through the origin and is orthogonal to the planes

$$\begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix} \bullet \mathbf{p} = 2 \quad \text{and} \quad \begin{pmatrix} 4 \\ 2 \\ 0 \end{pmatrix} \bullet \mathbf{p} = 1$$

[3 marks]

The normal of the plane must be orthogonal to the normals of the given planes. Furthermore the distance of the plane to the origin is zero. Hence:

$$\mathbf{n} = \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix} \times \begin{pmatrix} 4 \\ 2 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ -4 \\ -6 \end{pmatrix}, \; d = 0$$

B. Given is a point $\mathbf{q}_0$ and a vector $\mathbf{v}$ defining a line $\mathbf{q}(t)=\mathbf{q}_0+t\cdot\mathbf{v}$ and a vector $\mathbf{n}$ and scalar value d defining a plane $\mathbf{n}.\mathbf{p}=d$.

Derive the parametric equation of the line obtained by projecting the line $\mathbf{q}(t)=\mathbf{q}_0+t\cdot\mathbf{v}$ onto the given plane. Your equation should be dependent only on $\mathbf{q}_0$, $\mathbf{v}$, $\mathbf{n}$ and d. Clearly explain the different steps of your derivation.

[7 marks]

First we have to compute a point on the projected line – one such point is the intersection point of the line with the plane. The plane and line intersect if and only if

$$\mathbf{n}\bullet(\mathbf{q}_0+t_{\text{int}}\mathbf{v})=d \Leftrightarrow t_{\text{int}}=\frac{d-\mathbf{n}\bullet\mathbf{q}_0}{\mathbf{n}\bullet\mathbf{v}}$$

Hence the intersection point is

$$\mathbf{q}_{onPlane}=\mathbf{q}_0+t_{\text{int}}\mathbf{v}=\mathbf{q}_0+\frac{d-\mathbf{n}\bullet\mathbf{q}_0}{\mathbf{n}\bullet\mathbf{v}}\mathbf{v}$$

The direction of the line is equal to the direction of $\mathbf{v}$ projected onto the plane. This is

$$\mathbf{v}_{proj}=\mathbf{v}-\frac{\mathbf{v}\bullet\mathbf{n}}{\mathbf{n}\bullet\mathbf{n}}\mathbf{n}$$

Hence the projected line is $\mathbf{p}(t)=\mathbf{q}_{onPlane}+t\mathbf{v}_{proj}=\mathbf{q}_0+t_{\text{int}}\mathbf{v}=\left(\mathbf{q}_0+\frac{d-\mathbf{n}\bullet\mathbf{q}_0}{\mathbf{n}\bullet\mathbf{v}}\mathbf{v}\right)+t\left(\mathbf{v}-\frac{\mathbf{v}\bullet\mathbf{n}}{\mathbf{n}\bullet\mathbf{n}}\mathbf{n}\right)$

Remarks:

Note that the direction of the projected line can also be computed as $\mathbf{v}_{proj}=(\mathbf{v}\times\mathbf{n})\times\mathbf{n}$

Furthermore note that the computation of $\mathbf{q}_{onPlane}$ doesn't work if the line is parallel to the plane. A more general method to find a point on the plane is to project $\mathbf{q}_0$ onto the plane. This is done by computing its distance to the plane ($\frac{\mathbf{q}_0\bullet\mathbf{n}-d}{|\mathbf{n}|}$) and then subtracting from it a vector in normal direction which has this distance as length, i.e.

$$\mathbf{q}_{onPlane}=\mathbf{q}_0-\left(\frac{\mathbf{q}_0\bullet\mathbf{n}-d}{|\mathbf{n}|}\right)\frac{\mathbf{n}}{|\mathbf{n}|}=\mathbf{q}_0-\frac{\mathbf{q}_0\bullet\mathbf{n}-d}{\mathbf{n}\bullet\mathbf{n}}\mathbf{n}$$

Alternative solution:

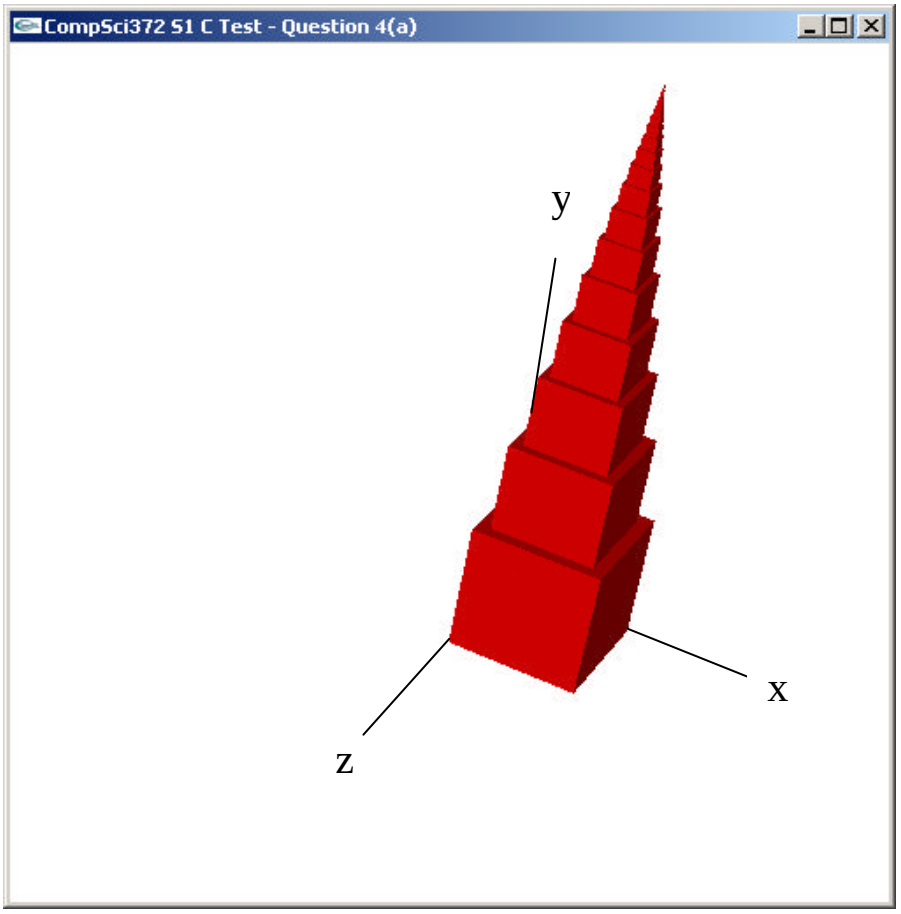
A very elegant solution is to project each point $\mathbf{q}(t)$ onto the plane. This is achieved by subtracting from it a vector in normal direction with a length equal to the distance of that point to the plane:

$$\mathbf{p}(t)=\mathbf{q}(t)-\frac{\mathbf{q}(t)\bullet\mathbf{n}-d}{\mathbf{n}\bullet\mathbf{n}}\mathbf{n}=(\mathbf{q}_0+t\mathbf{v})-\frac{(\mathbf{q}_0+t\mathbf{v})\bullet\mathbf{n}-d}{\mathbf{n}\bullet\mathbf{n}}\mathbf{n}$$

Rearranging terms shows that this solution is equivalent to the solution in the above remarks.

Question 4 – Transformations [10 marks]

A. Given is a function *drawCube()* which draws an axis-aligned **unit cube** with side length 1 **centred** at the origin.
Use this function to complete the display method on the next page so that it draws the object shown in the image below. The bottom part of the object is a unit cube (side length one) located in the corner of the first quadrant of the xz-plane. The whole object consists of *nCubes* cubes where each cube is centred on top of the cube below it and where each cube has a side length of 0.8 times the side length of the cube below it. [6 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);

    int nCubes=100;
    // Draw the scene
```

```
float scale=1;
float shiftY=0;
for(int i=1;i<nCubes;i++)
{
  glPushMatrix();
  glTranslatef(0.5f, 0.5f+shiftY, 0.5);
  glScalef(scale, scale, scale);
  drawCube();
  glPopMatrix();
  shiftY+=scale*0.9f;
  scale*=0.8f;
}
```

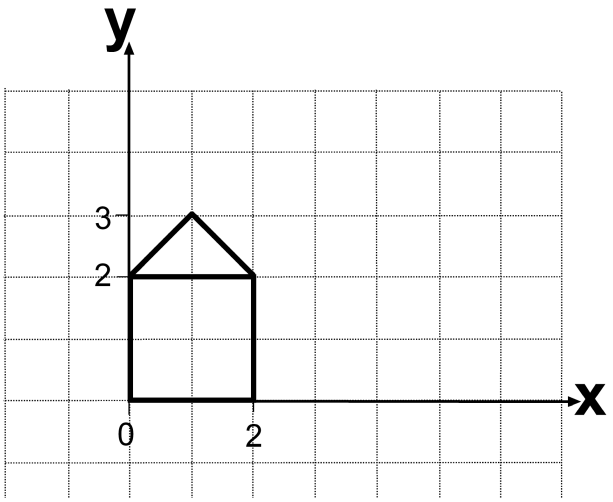
Note: each cube is on top of the previous cube, hence a cube is translated in y-direction by half the height of the old cube plus half the height of the new cube. Since scale gives the height of the current cube the new cube is $\frac{scale}{2}+\frac{scale*0.8}{2}=scale*0.9$ units above the previous cube.

Here is a more efficient solution (which is a bit harder to understand):

```
float scale=1;
for(int i=1;i<nCubes;i++)
{
  glPushMatrix();
  glTranslatef(0.5f, 0.5f, 0.5);
  glScalef(scale, scale, scale);
  drawCube();
  glPopMatrix();
  glTranslatef(0, scale*0.9, 0);
  scale*=0.8f;
}
```

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

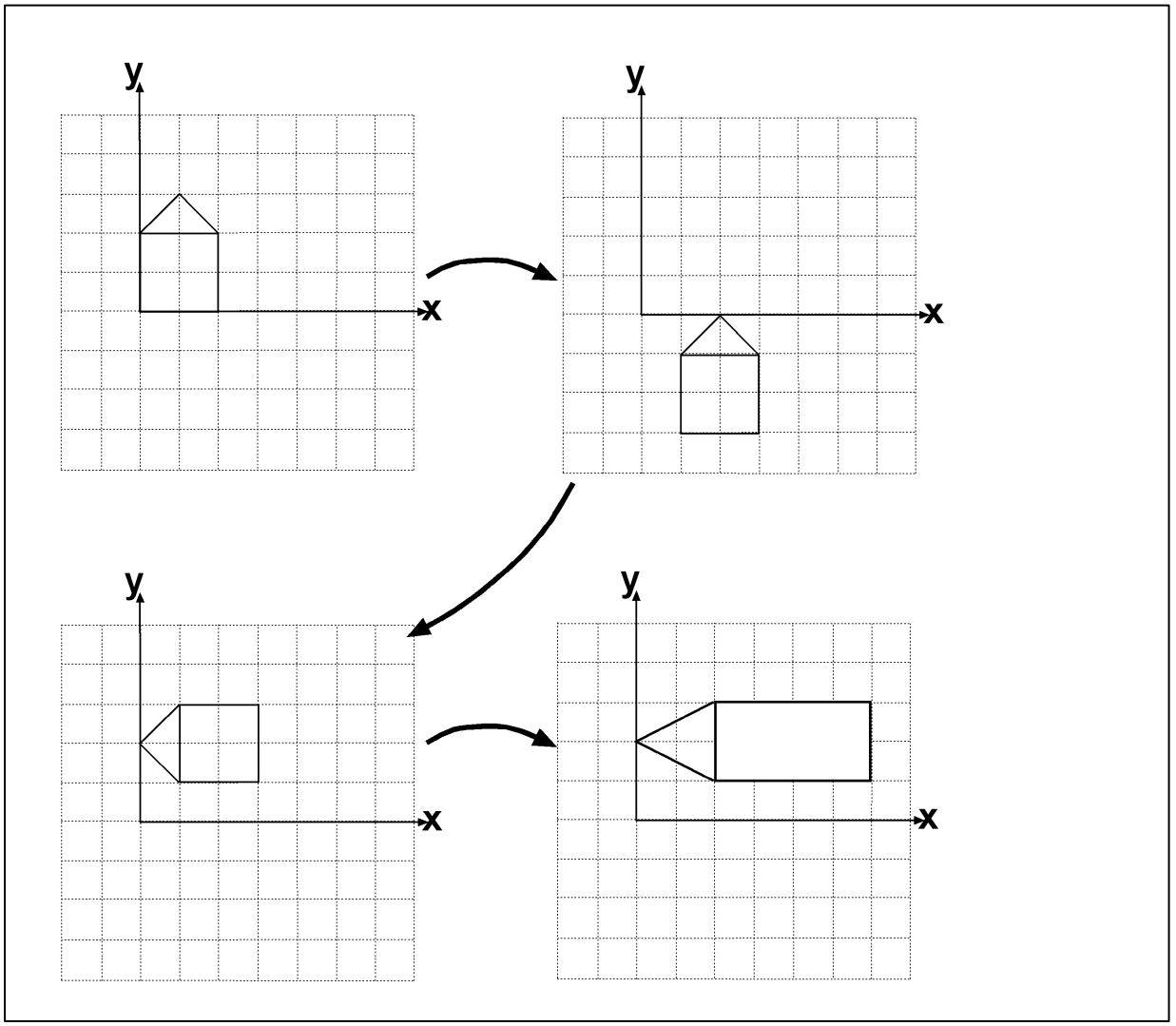
B. Given is a 2D house-shaped object shown on the right and a homogeneous transformation matrix **M** where



$$\mathbf{M} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos 90^\circ & -\sin 90^\circ & 0 \\ \sin 90^\circ & \cos 90^\circ & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & -3 \\ 0 & 0 & 1 \end{pmatrix}$$

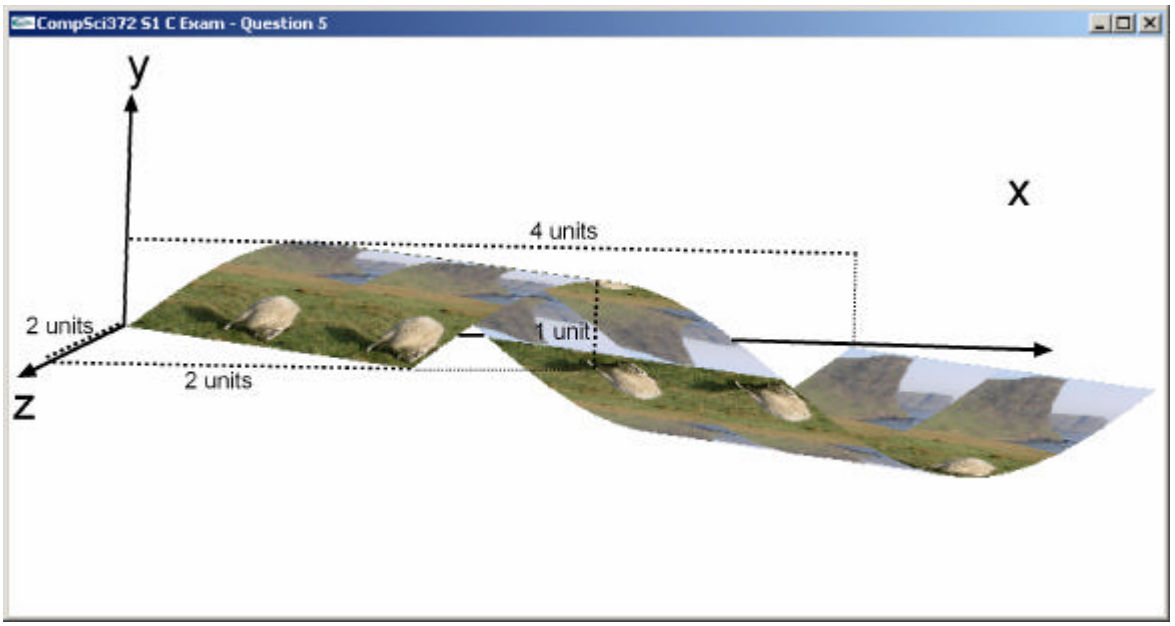
Draw in the answer space below the house-shaped object after transforming it with **M**.

[4 marks]



Question 5 – Modelling and texture mapping [10 marks]

A. Determine the parametric equation **p**(s,t)=(x(s,t),y(s,t),z(s,t)) of the extrusion surface shown in the image below. The surface is obtained by extruding a sine curve (which starts at the point (0,0,0) and ends at the point (4,0,0)) along the vector (2,0,2). [4 marks]



The since curve from point (0,0,0) to point (4,0,0) is given by the equation
$\mathbf{c}(t) = (4t, \sin 2\pi t, 0)$
Extruding this curve along the vector (2,0,2) gives the surface $\mathbf{p}(s,t) = (4t + 2s, \sin 2\pi t, 2s)$
where 0=s,t=1.

B. Assume the parametric equation of the surface in part (A) is **p**(*s*,*t*) where 0=*s*,*t*=1. Assume the parameter *t* specifies the original sine curve and the parameter *s* the extrusion direction.

Given is a function

```
float* point(float s, float t)
```

which implements **p**(*s*,*t*) and returns an array of three floats.

Complete the code below which draws the textured surface shown in (A). The texture image (shown on the right) is repeated four times in the *t* parameter direction and two times in the *s* parameter direction [6 marks].

[Hint: The prototype of the function for specifying a 2d texture coordinate is

```
void glTexCoord2f(float s, float t);
```
]

```
const int NUM_STEPS_T=32;
const int NUM_STEPS_S=16;

void display(void)
{
   // setting up the camera
   glMatrixMode( GL_MODELVIEW );  // Set the view matrix ...
   glLoadIdentity();              // ... to identity.
   gluLookAt(0,0,50, 2.5,0,0, 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);

   // selecting the texture and enabling texture mapping.
   // NOTE: Assume that the texture wrap parameters are set to GL_REPEAT.

   glBindTexture(GL_TEXTURE_2D, texName);
   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);

   glEnable(GL_TEXTURE_2D);
```

```
// draw the surface and with suitable texture coordinates

float s,t1,t2;
for (int j=0; j< NUM_STEPS_T; j++)
{
  t1=(float) j/(float) NUM_STEPS_T;
  t2=(float) (j+1)/(float) NUM_STEPS_T;
  glBegin(GL_QUAD_STRIP);
  for (int i=0; i<= NUM_STEPS_S; i++)
  {
    s=(float) i/(float) NUM_STEPS_S;
    glTexCoord2f(2*s,4*t1);
    glVertex3fv(point(s,t1));
    glTexCoord2f(2*s,4*t2);
    glVertex3fv(point(s,t2));
  }
  glEnd();
}
```

```
  glDisable(GL_TEXTURE_2D);
  glFlush ();
  glutSwapBuffers();
}
```