Computer Science

COMPSCI 372 S2 C – Exercise Sheet 6

22nd August 2008

Q1: Look at the code of the "Colour Cube" example in the handout "Modelling with Polygonal Meshes".

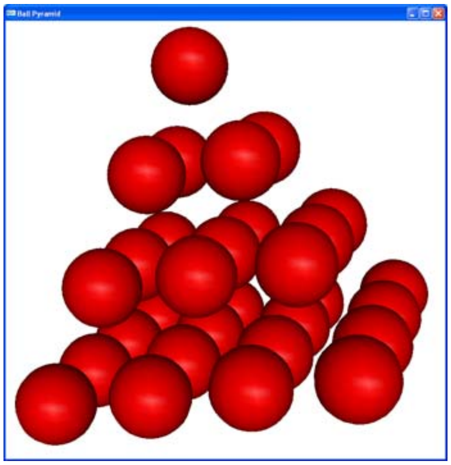
(a) In which order are the faces of the cube drawn on the screen (list the faces by their colours)

(b) What picture do you get if the camera is at the point (3,3,3) and looks towards the origin?

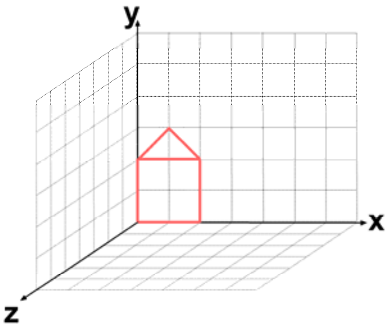
(c) What picture do you get if you use the same camera set-up as in (b), but you disable depth testing? Explain why the picture looks this way.

Q2: Given is a triangle with the vertices $A = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$, $B = \begin{pmatrix} 6 \\ 0 \\ 0 \end{pmatrix}$, $C = \begin{pmatrix} 0 \\ 4 \\ 0 \end{pmatrix}$ and the vertex colours C_A=(1,0,0), C_B=(0,1,0), C_A=(0,0,1). What is the colour at the point $P = \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}$?

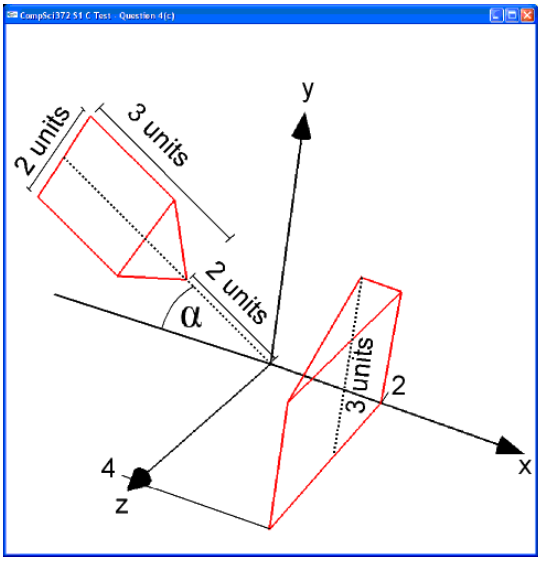
Q3: Write a display method which generates a pyramid of balls as shown in the image on the right.



Q4: Give is a function drawHouse() which draws a wire frame house in the xy-plane with width 2 and height 3 as shown in the image on the right.



Use the function `drawHouse()` and OpenGL transformations in order to draw the scene show in the image on the right. Assume you have a variable `alpha` which defines the angle α..

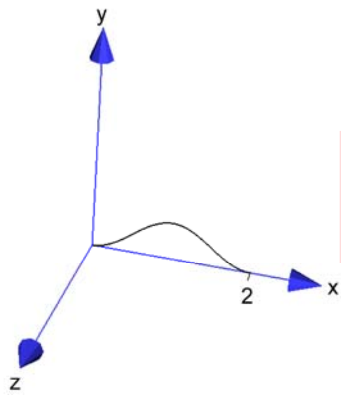


```
void display(void)
{
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    gluLookAt(0,0,20, 0,1.25,0, 0,1,0);
    trackball.tbMatrix();
    glClear( GL_COLOR_BUFFER_BIT |
    GL_DEPTH_BUFFER_BIT);
    float alpha=someValue;    // assume this value is defined elsewhere in the code



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

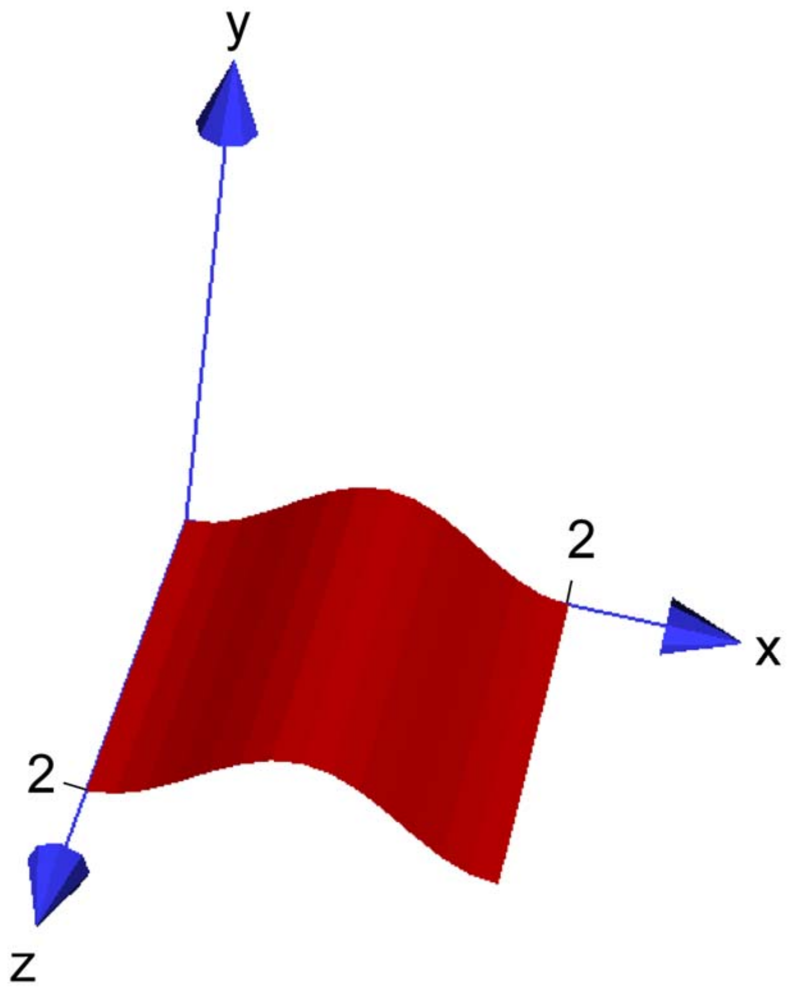
Q5: Given is a the function

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

which returns an array of three floats representing the points of the parametric curve shown in the image on the right. The parameter *t* of the function is 0≤*t*≤1 and *t*=0 gives the curve point at the origin of the coordinate system.



Write a function for displaying the surface shown in the image below. The profile of the surface is the above curve. **Note that the surface is flat shaded and you have to compute for every polygon of the surface one surface normal**.
You are allowed to use the functions and classes in appendix A.



```
void display(void)
{
      glMatrixMode( GL_MODELVIEW ); // Set the view matrix ...
      glLoadIdentity();             // ... to identity.
      gluLookAt(0,5,20, 1,0,0, 0,1,0); // camera is on the z-axis
      trackball.tbMatrix();         // use a trackball to rotate scene

      glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

      glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE,
                   mat_ambient_and_diffuse);
      glShadeModel(GL_FLAT);

      // Draw the surface
      int nSteps=15;    // number of steps for subdividing the curve
```

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

Appendix A

```
class CVec3df {
       public:
           // Constructors/ Destructor
           CVec3df();
           CVec3df(float x, float y, float z);
               CVec3df(const CVec3df& v); // Copy constructor
               virtual ~CVec3df();

           // Assignment operator
           CVec3df& operator=(const CVec3df& v1);

               // Vector in array form
           float* getArray() { return v;}

           // some other operators
           CVec3df& operator+=(const CVec3df& v1);
           CVec3df& operator-=(const CVec3df& v1);
           CVec3df& operator*=(float scalar);
           CVec3df& operator/=(float scalar);

           friend CVec3df operator+(const CVec3df& v1, const CVec3df& v2);
           friend CVec3df operator-(const CVec3df& v1, const CVec3df& v2);
           friend CVec3df operator*(float scalar, const CVec3df& v1);
           friend CVec3df operator*(const CVec3df& v1, float scalar);
           friend CVec3df operator/(const CVec3df& v1, float scalar);
           friend CVec3df operator*(const CVec3df& v1, const CVec3df& v2);
           friend CVec3df operator-(const CVec3df& v1);
           friend bool operator==(const CVec3df& v1, const CVec3df& v2);
           friend bool operator!=(const CVec3df& v1, const CVec3df& v2);

           // normalize
           CVec3df normalise(void) const;      // returns a normalised vector
           void normaliseDestructive(void);    // normalises vector object

           float dot(const CVec3df& v1) const;        // dot product
           CVec3df cross(const CVec3df& v1) const;    // cross product
       private:
           float* v;
};

// more convenient way to use the dot and cross products
inline float dot(const CVec3df& v1, const CVec3df& v2) { return v1.dot(v2); }
inline CVec3df cross(const CVec3df& v1, const CVec3df& v2) { return
v1.cross(v2); }
```