

Letterkenny Institute of Technology

BSc in Computing (Games Development)

Subject: Games Programming 2

Level: 7

Date: Autumn 2007

Examiner:

Dr. J.G. Campbell

Dr. M.D.J. McNeill

Time Allowed: Three hours.

INSTRUCTIONS

Answer **four** questions from six. Failure to show derivation of answers and/or calculations may result in loss of marks.

1. OpenGL primitives and interaction.

The code in Figures 3 and 4 draws a the line figure shown in Figure 1; keyboard and mouse interaction can be used to modify the position and size of the rectangle.

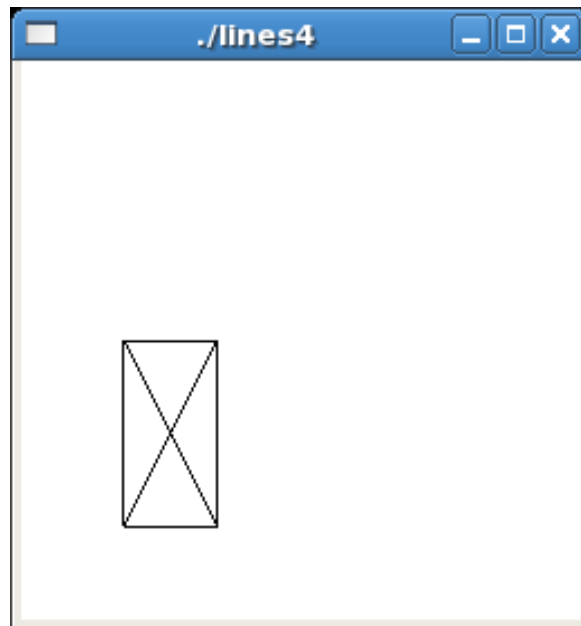


Figure 1: Line figure.

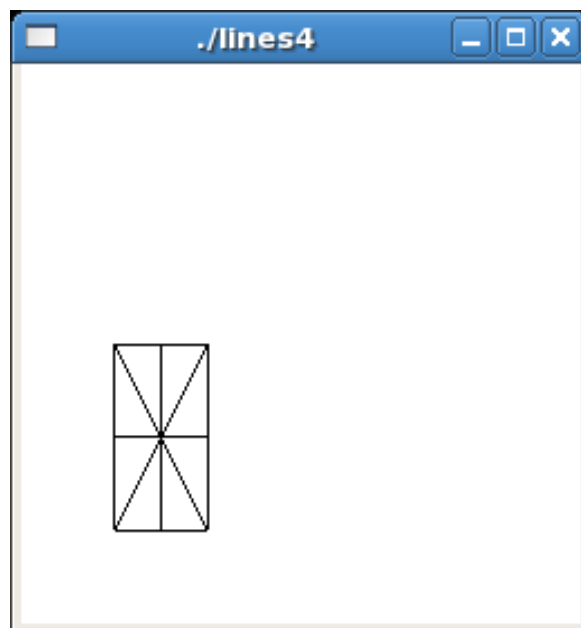


Figure 2: Line figure with added vertical and horizontal lines.

In Appendix E, Figure 17, shows the list of possible arguments for `glBegin` and Figure 18 gives a diagrammatic explanation.

(a) Explain each the eight lines marked `/* a */` to `/* f */` in Figures 3 and 4.

[9 marks]

(b) Explain precisely the circumstances in which `display` gets executed.

[2 marks]

(c) Rewrite the code which draws the outer part of the rectangle (first eight calls to `glVertex2d`) using `GL_LINE_STRIP`, i.e. so that repeated (duplicate) specifications of vertexes are not needed.

[4 marks]

(d) Write code with explanation which draws the vertical and horizontal lines shown in Figure 2.

[4 marks]

(e) Give an outline of the additional code that would be necessary to rotate the figure under control of the keyboard key 'R'. (i) what additional variable(s) will need to be declared and where? (ii) what code will need to be added to keyboard; (iii) what code will need to be added elsewhere to carry out the rotation?

[6 marks]

```

#include <GL/glut.h> #include <stdlib.h> #include <stdio.h>
double x = 0.0, y = 0.0, size = 1.0;
void display(void){
    glClear(GL_COLOR_BUFFER_BIT); /* a */
    glColor3f(0.0, 0.0, 0.0); /* b */
    glBegin(GL_LINES); /* c */
        glVertex2d(x + 0.0*size, y + 2.0*size); /* d */
        glVertex2d(x + 1.0*size, y + 2.0*size);
        glVertex2d(x + 1.0*size, y + 2.0*size);
        glVertex2d(x + 1.0*size, y + 0.0*size);
        glVertex2d(x + 1.0*size, y + 0.0*size);
        glVertex2d(x + 0.0*size, y + 0.0*size);
        glVertex2d(x + 0.0*size, y + 0.0*size);
        glVertex2d(x + 0.0*size, y + 2.0*size);

        glVertex2d(x + 0.0*size, y + 0.0*size);
        glVertex2d(x + 1.0*size, y + 2.0*size);

        glVertex2d(x + 0.0*size, y + 2.0*size);
        glVertex2d(x + 1.0*size, y + 0.0*size);
    glEnd();
    glFlush();
}
void init(void) {
    glClearColor (1.0, 1.0, 1.0, 0.0); /* e */
    glShadeModel (GL_FLAT);
}
void reshape(int w, int h){
    glViewport (0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode(GL_PROJECTION);    glLoadIdentity();
    glOrtho(-1.0, 5.0, -1.0, 5.0, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);    glLoadIdentity();
}
void mouse(int button, int state, int xm, int ym){
    switch (button) {
        case GLUT_LEFT_BUTTON:
            if (state == GLUT_DOWN)x+= 0.1;
            break;
        case GLUT_MIDDLE_BUTTON:
            if (state == GLUT_DOWN)y+= 0.1;
            break;
        case GLUT_RIGHT_BUTTON:
            if (state == GLUT_DOWN)size+= 0.1;
            break;
        default:            break;
    }
    glutPostRedisplay();
}

```

Figure 3: Lines4.c part 1

```

void keyboard(unsigned char key, int xkb, int ykb){
    switch (key) {
        case 27: /*ESCAPE*/
        case 'Q':
        case 'q':
            exit(0);
            break;
        case 'x':
            x = x - 0.1;
            break;
        case 'X':
            x = x + 0.1;
            break;
        case 'y':
            y = y - 0.1;
            break;
        case 'Y':
            y = y + 0.1;
            break;
        case 's':
            size = size - 0.1;
            break;
        case 'S':
            size = size + 0.1;
            break;
    }
    glutPostRedisplay();
}

int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (250, 250);
    glutInitWindowPosition (100, 100);
    glutCreateWindow (argv[0]);
    init ();
    glutDisplayFunc(display); /* f */
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutMouseFunc(mouse);
    glutMainLoop();
    return 0;
}

```

Figure 4: Lines4.c part 2

2. OpenGL Modelling and Viewing and associated matrices.

(a) The code in Figures 6, 7 and 8 draws the transformed figures shown in Figure 5 and prints the *modelview* matrices shown in Figure 9.

(i) Give a precise explanation the content of matrices:

```
GL_MODELVIEW_MATRIX 1,  
GL_MODELVIEW_MATRIX 2,  
GL_MODELVIEW_MATRIX 3 and  
GL_MODELVIEW_MATRIX 4.
```

[8 marks]

(ii) Give a precise explanation the content of matrices:

```
GL_MODELVIEW_MATRIX 6 and GL_MODELVIEW_MATRIX 7
```

[8 marks]

Ensure that you clearly explain the differences in them. *Hint: order of application of transformations.*

(c) Give a brief explanation of `glPushMatrix` and `glPopMatrix`.

[3 marks]

(d) What is the modelview matrix after each numbered call (`//1`, `// 2`, `//3`) in the following sequence?

```
glLoadIdentity ();  
glRotatef (45.0, 0.0, 0.0, 1.0); //1  
glRotatef (45.0, 0.0, 0.0, 1.0); //2  
glTranslatef (50.0, 70.0, 0.0); //3
```

You may write your answer in terms of numbers, or in terms of $\cos \theta$, t_x , t_y etc.

[6 marks]

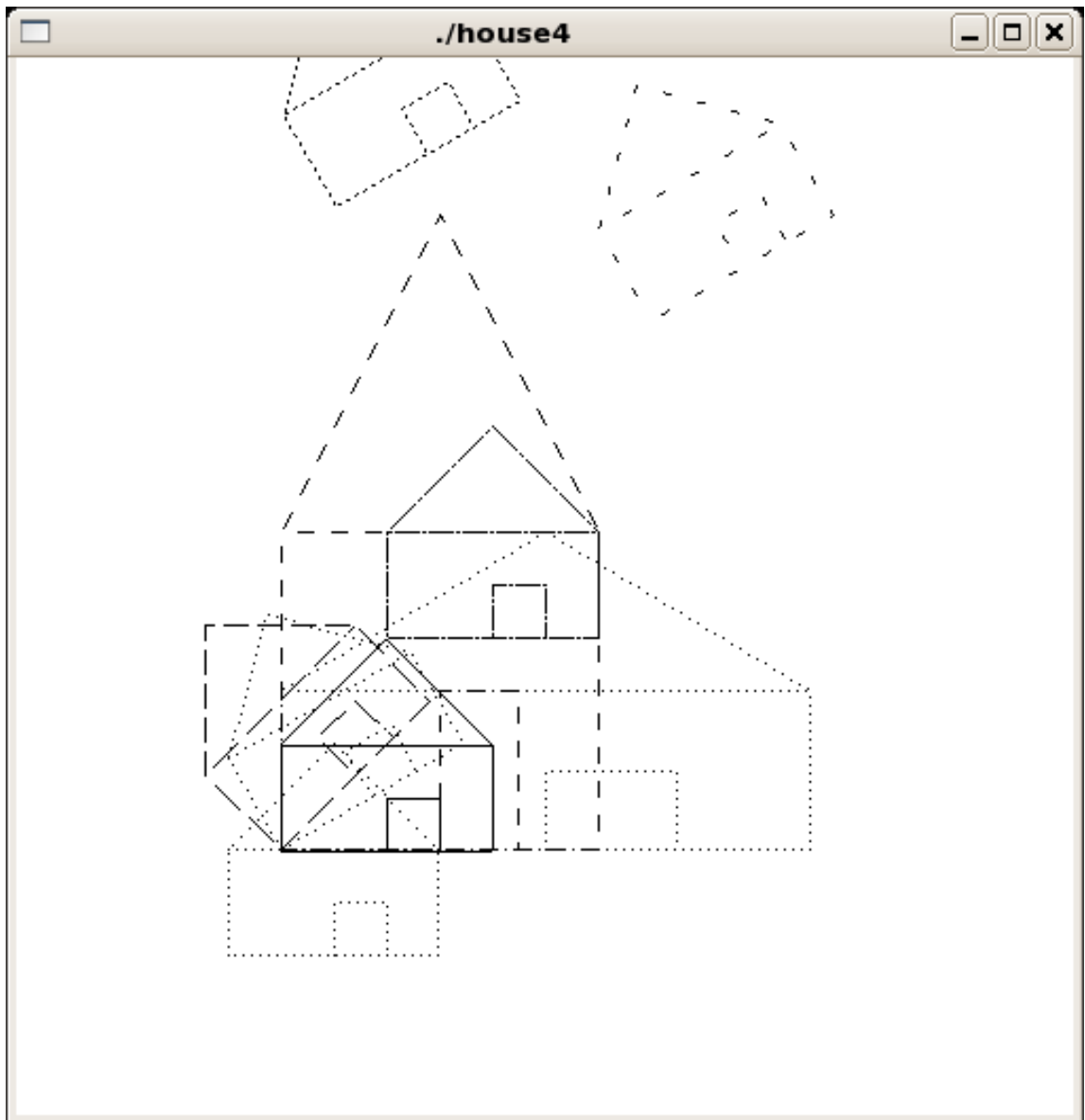


Figure 5: Graphical output from house4.c.

```

double hw= 20.0, hh = 10.0, hr=20.0;

void matPrint(int nmat, char *msg){
    double mat[16];
    int r, c; int nc= 4, nr= 4, i;

    glGetDoublev(nmat, mat);
    printf("%s\n", msg);
    printf("[");
    for(r= 0; r< nr; r++){
        printf("(");
        for(c= 0; c< nc; c++){
            i= r + c*nr;
            printf("%.4f", mat[i]);
            if(c == nc-1) printf("\n");
            else printf(", ");
        }
    }
    printf("]\n");
}

void init(void) {
    glClearColor (1.0, 1.0, 1.0, 0.0);
    glShadeModel (GL_FLAT);
}

void house(double width, double wallHt, double roofHt) {
    //left wall, ...
    glBegin(GL_LINE_STRIP);
        glVertex2d(0., 0.);
        glVertex2d(0., wallHt);
        glVertex2d(width/2., roofHt);
        glVertex2d(width, wallHt);
        glVertex2d(width, 0.);
        glVertex2d(0., 0.);
    glEnd();
    glBegin(GL_LINE_STRIP);
        glVertex2d(0., wallHt);
        glVertex2d(width, wallHt);
    glEnd();
    glBegin(GL_LINE_STRIP);
        glVertex2d(width/2., 0.);
        glVertex2d(width/2., wallHt/2.);
        glVertex2d(width*3./4., wallHt/2.);
        glVertex2d(width*3./4., 0.);
    glEnd();
}

```

Figure 6: house4.c part 1

```

void display(void){
    double mat1[16]= {1.5000, 0.0000, 0.0000, 0.0000,
                      0.0000, 3.0000, 0.0000, 0.0000,
                      0.0000, 0.0000, 1.0000, 0.0000,
                      0.0000, 0.0000, 0.0000, 1.0000};
    double mat2[16]= {0.707, 0.707, 0.0000, 0.0000,
                      -0.707, 0.707, 0.0000, 0.0000,
                      0.0000, 0.0000, 1.0000, 0.0000,
                      0.0000, 0.0000, 0.0000, 1.0000};
    double mat3[16]= {0.0000, 0.0000, 0.0000, 10.0000,
                      0.0000, 0.0000, 0.0000, 20.0000,
                      0.0000, 0.0000, 1.0000, 0.0000,
                      0.0000, 0.0000, 0.0000, 1.0000};
    glClear (GL_COLOR_BUFFER_BIT);   glColor3f (0.0, 0.0, 0.0);

    glLoadIdentity ();
    matPrint(GL_MODELVIEW_MATRIX, "GL_MODELVIEW_MATRIX 1");
    house(hw, hh, hr);

    glEnable (GL_LINE_STIPPLE);
    glLineStipple (1, 0x1111);
    glLoadIdentity ();   glTranslatef (-5.0, -10.0, 0.0);
    matPrint(GL_MODELVIEW_MATRIX, "GL_MODELVIEW_MATRIX 2");
    house(hw, hh, hr);

    glLineStipple (1, 0x1111);
    glLoadIdentity ();   glScalef (2.5, 1.5, 1.0);
    matPrint(GL_MODELVIEW_MATRIX, "GL_MODELVIEW_MATRIX 3");
    house(hw, hh, hr);

    glLineStipple (1, 0x1111);
    glLoadIdentity ();   glRotatef (30.0, 0.0, 0.0, 1.0);
    matPrint(GL_MODELVIEW_MATRIX, "GL_MODELVIEW_MATRIX 4");
    house(hw, hh, hr);

    glLineStipple (1, 0x6666);   glLoadIdentity ();
    glRotatef (30.0, 0.0, 0.0, 1.0);
    matPrint(GL_MODELVIEW_MATRIX, "GL_MODELVIEW_MATRIX 5");
    glTranslatef (35.0, 50.0, 0.0);
    matPrint(GL_MODELVIEW_MATRIX, "GL_MODELVIEW_MATRIX 6");
    house(hw, hh, hr);

    glLineStipple (1, 0xF000);   glLoadIdentity ();
    glTranslatef (35.0, 50.0, 0.0);
    glRotatef (30.0, 0.0, 0.0, 1.0);
    matPrint(GL_MODELVIEW_MATRIX, "GL_MODELVIEW_MATRIX 7");
    house(hw, hh, hr);
}

```

Figure 7: house4.c part 2

```

glLineStipple (1, 0xFF00);
glLoadIdentity ();
glMultMatrixd(mat1);
matPrint(GL_MODELVIEW_MATRIX, "GL_MODELVIEW_MATRIX 8");
house(hw, hh, hr);

glLineStipple (1, 0xFFFF0);
glLoadIdentity ();
glMultMatrixd(mat2);
matPrint(GL_MODELVIEW_MATRIX, "GL_MODELVIEW_MATRIX 9");
house(hw, hh, hr);

glLineStipple (1, 0xFFFF);
glLoadIdentity ();
glMultMatrixd(mat3);
matPrint(GL_MODELVIEW_MATRIX, "GL_MODELVIEW_MATRIX 10");
house(hw, hh, hr);

glDisable (GL_LINE_STIPPLE);
glFlush ();
}

```

Figure 8: house4.c part 3

```

GL_MODELVIEW_MATRIX 1
[(1.0000, 0.0000, 0.0000, 0.0000)
(0.0000, 1.0000, 0.0000, 0.0000)
(0.0000, 0.0000, 1.0000, 0.0000)
(0.0000, 0.0000, 0.0000, 1.0000)]
GL_MODELVIEW_MATRIX 2
[(1.0000, 0.0000, 0.0000, -5.0000)
(0.0000, 1.0000, 0.0000, -10.0000)
(0.0000, 0.0000, 1.0000, 0.0000)
(0.0000, 0.0000, 0.0000, 1.0000)]
GL_MODELVIEW_MATRIX 3
[(2.5000, 0.0000, 0.0000, 0.0000)
(0.0000, 1.5000, 0.0000, 0.0000)
(0.0000, 0.0000, 1.0000, 0.0000)
(0.0000, 0.0000, 0.0000, 1.0000)]
GL_MODELVIEW_MATRIX 4
[(0.8660, -0.5000, 0.0000, 0.0000)
(0.5000, 0.8660, 0.0000, 0.0000)
(0.0000, 0.0000, 1.0000, 0.0000)
(0.0000, 0.0000, 0.0000, 1.0000)]
GL_MODELVIEW_MATRIX 5
[(0.8660, -0.5000, 0.0000, 0.0000)
(0.5000, 0.8660, 0.0000, 0.0000)
(0.0000, 0.0000, 1.0000, 0.0000)
(0.0000, 0.0000, 0.0000, 1.0000)]
GL_MODELVIEW_MATRIX 6
[(0.8660, -0.5000, 0.0000, 5.3109)
(0.5000, 0.8660, 0.0000, 60.8013)
(0.0000, 0.0000, 1.0000, 0.0000)
(0.0000, 0.0000, 0.0000, 1.0000)]
GL_MODELVIEW_MATRIX 7
[(0.8660, -0.5000, 0.0000, 35.0000)
(0.5000, 0.8660, 0.0000, 50.0000)
(0.0000, 0.0000, 1.0000, 0.0000)
(0.0000, 0.0000, 0.0000, 1.0000)]
GL_MODELVIEW_MATRIX 8
[(1.5000, 0.0000, 0.0000, 0.0000)
(0.0000, 3.0000, 0.0000, 0.0000)
(0.0000, 0.0000, 1.0000, 0.0000)
(0.0000, 0.0000, 0.0000, 1.0000)]
GL_MODELVIEW_MATRIX 9[(0.7070, -0.7070, 0.0000, 0.0000)
(0.7070, 0.7070, 0.0000, 0.0000)
(0.0000, 0.0000, 1.0000, 0.0000)
(0.0000, 0.0000, 0.0000, 1.0000)]
GL_MODELVIEW_MATRIX 10[(1.0000, 0.0000, 0.0000, 10.0000)
(0.0000, 1.0000, 0.0000, 20.0000)
(0.0000, 0.0000, 1.0000, 0.0000)
(0.0000, 0.0000, 0.0000, 1.0000)]

```

Figure 9: house4.c output

3. OpenGL lighting.

Eqn. 1 summarises the version of the Phong lighting model used by OpenGL,

$$I_{total} = m_e + \sum_{i=1}^{N_l} sa^i att^i [l_a^i m_a + l_d^i m_d \max(\mathbf{N} \cdot \mathbf{L}^i, 0) + l_s^i m_s \max((\mathbf{V} \cdot \mathbf{R}^i), 0)^s]. \quad (1)$$

\mathbf{N} is the normal to the surface at the vertex in question; \mathbf{L}^i is the direction of light i ; \mathbf{V} is the viewing (eye) direction and \mathbf{R}^i is the specular reflection direction for light i ; sa is the spotlight attenuation factor and att distance attenuation; \mathbf{R} is given by eqn. 2.

$$\mathbf{R} = 2(\mathbf{N} \cdot \mathbf{L})\mathbf{N} - \mathbf{L}. \quad (2)$$

The geometry involved is shown in Figure 10.

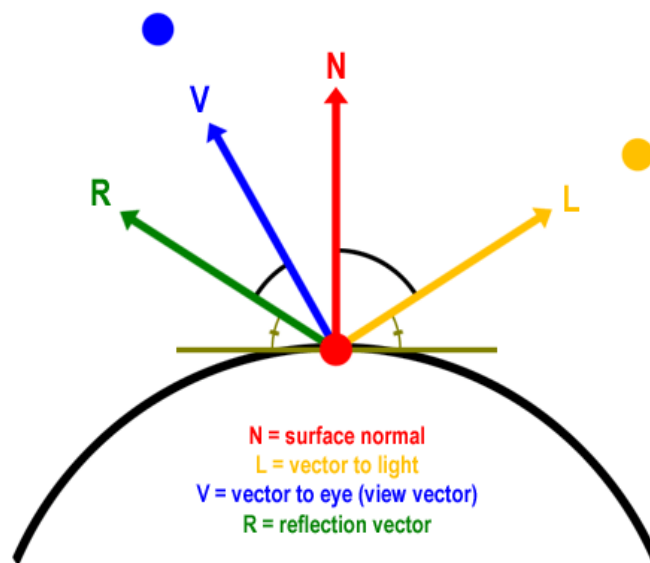


Figure 10: Phong lighting model; L is the direction of the light; N is the normal to the surface, V is the direction of the viewer (eye); R is the direction where reflection angle is equal to the incidence (lighting) angle.

(a) Explain how *colour* enters eqn. 1

[5 marks]

(b) Based on eqn. 1, explain the following types of lighting, both lighting source and material:

- (i) *Ambient*;
- (ii) *Diffuse*;
- (iii) *Specular*;
- (iv) *Emissive*.

[12 marks]

- (c) Given the code fragment below, and, **considering ambient and diffuse only**, explain what will be the colour (shade) at vertex `vf[0]`. Assume that the light direction is in the direction of the normal; **no attenuation**. You should carefully explain your calculations. Your answer should be an RGB triple.

[5 marks]

```
GLfloat l_amb[] = { 0.2, 0.3, 0.8, 1.0 };
GLfloat l_dif[] = { 0.4, 0.5, 0.9, 1.0 };
GLfloat l_spc[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat m_spc[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat m_spc[] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat m_shn[] = { 5.0 };
GLfloat m_amb[] = { 0.5, 0.5, 0.5, 1.0 };
GLfloat m_dif[] = { 0.8, 0.8, 0.8, 1.0 };

GLfloat l_pos[] = { 20.0, 20.0, 20.0, 0.0 };
glLightfv (GL_LIGHT0, GL_POSITION, l_pos);
glLightfv (GL_LIGHT0, GL_DIFFUSE, l_dif);
glLightfv (GL_LIGHT0, GL_AMBIENT, l_amb);
glLightfv (GL_LIGHT0, GL_SPECULAR, l_spc);

glMaterialfv(GL_FRONT, GL_SPECULAR, m_spc);
glMaterialfv(GL_FRONT, GL_SHININESS, m_shn);
glMaterialfv(GL_FRONT, GL_AMBIENT, m_amb);
glMaterialfv(GL_FRONT, GL_DIFFUSE, m_dif);

glVertex3fv(vf[0]);
```

- (d) Now, consider only *specular reflection* and assume that the viewing direction (**V**) is in the same direction as the specular reflection direction (**R**) so that $\mathbf{V} \cdot \mathbf{R} = 1$, what will be the colour of the light reflected, given the following code? Answer with an RGB triple.

```
GLfloat l_spc[] = { 1.0, 0.5, 0.25, 1.0 };
GLfloat m_spc[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat m_shn[] = { 2.0 };
```

[3 marks]

4. **Pointers, arrays, heap memory, dangling pointers, garbage, memory leaks.**

The program in Figures 11 and 12 shows examples of global, local and heap variables.

- (a) Briefly explain the lifetimes of each of variables `g`, `x`, and `a`. [6 marks]
- (b) Briefly explain the lifetime of the heap array referenced by `p`. [2 marks]
- (c) The heap array referenced by `p` remains in existence after `p` has been destroyed. Explain. [2 marks]
- (d) Lines `// A` and `// B` have the same effect. Explain the similarity between array indexing `[]` and pointer dereferencing `*`. [4 marks]
- (e) In function `parrr` we have a pointer to an array being returned (perfectly correctly and legally). On the other hand the same in `badarr`, although it compiles and executes, is very *incorrect* (compare the printout of `py` with that of `p2`). Explain. [3 marks]
- (f) At line `// C` the original heap array becomes *garbage*. Explain. [3 marks]
- (g) If the problem mentioned in (f) occurred repeatedly in a large program that runs for a long time (e.g. a web server), explain what serious problem might eventually cause the program to crash? [3 marks]
- (h) Write the code that will remove the problem mentioned in (f). [2 marks]

```

//----- tarr3.cpp -----
#include <iostream>
using namespace std;

const int n= 5;

double g[n]= {0,1,2,3,4};

void arr(){
    cout<< "In arr\n";
    double x[n]= {10.0,11.1,12.2,13.3,14.4};
    for(int i= 0; i< n; i++)cout<< x[i]<< " ";
    cout<< endl;
    double *px = &x[0];
    for(int i= 0; i< n; i++)cout<< *(px+i)<< " ";
    cout<< endl;
}

double *badarr(){
    cout<< "In badarr\n";
    double y[n]= {110.0,111.1,112.2,113.3,114.4};
    cout<< "y = ";
    for(int i= 0; i< n; i++)cout<< y[i]<< " ";
    cout<< endl;
    double *py = &y[0];
    cout<< "py = ";
    for(int i= 0; i< n; i++)cout<< *(py+i)<< " ";
    cout<< endl;
    return py;
}

double *parr(int m){
    cout<< "In parr\n";
    double *p= new double[m];
    for(int i= 0; i< m; i++)p[i]= double(i+20);
    cout<< "p = ";
    for(int i= 0; i< m; i++)cout<< p[i]<< " "; // A
    cout<< endl;
    cout<< "p = ";
    for(int i= 0; i< m; i++)cout<< *(p+i)<< " "; // B
    cout<< endl;
    return p;
} //... continued

```

Figure 11: Global, local and heap variables

```

int main(){
    cout<< "In main\n";
    double a[n]= {0.1,1.2,2.3,3.4,4.5};

    cout<< "a = ";
    for(int i= 0; i< n; i++)cout<< a[i]<< " ";
    cout<< endl;

    arr();

    int length = 6;
    double *p1= parr(length);

    cout<< "p1 = ";
    for(int i= 0; i< length; i++)cout<< p1[i]<< " ";
    cout<< endl;

    p1 = new double[n]; // C
    for(int i= 0; i< n; i++)p1[i]= double(i+100);
    cout<< "p1 = ";
    for(int i= 0; i< n; i++)cout<< p1[i]<< " ";
    cout<< endl;

    double *p2= badarr();

    cout<< "p2 = ";
    for(int i= 0; i< n; i++)cout<< p2[i]<< " ";
    cout<< endl;

    return 0;
}

```

Figure 12: Global, local and heap variables, part 2

```
In main
a = 0.1 1.2 2.3 3.4 4.5
In arr
10 11.1 12.2 13.3 14.4
10 11.1 12.2 13.3 14.4
In parr
p = 20 21 22 23 24 25
p = 20 21 22 23 24 25
p1 = 20 21 22 23 24 25
p1 = 100 101 102 103 104
In badarr
y = 110 111.1 112.2 113.3 114.4
py = 110 111.1 112.2 113.3 114.4
p2 = 110 110 5.68837e-315 2.122e-314 114.4
```

Figure 13: Output from tarr3.cpp

5. [Header files, programs with multiple compilation units, declaration of functions (prototypes), pass-by-value ...]

A complete program `Tr5b.cpp` is shown in Figure 14, together with its output.

- (a) Explain precisely what is the purpose of `#include <iostream>`.

[4 marks]

- (b) Explain precisely what is the purpose of `using namespace std;`.

[4 marks]

- (c) We have introduced a small compilation error, so that the compiler complains:

```
Tr5b.cpp:23: error: nl was not declared in this scope
```

Explain the nature of the error and how to correct it; hint: add something at `// A`

[4 marks]

- (d) Explain the difference between (i) the compilation stage and (ii) the linking stage, when compiling and linking a C++ program.

[4 marks]

- (e) We have now split the program into a main program file (shown in Figure 15), and functions files, `funcs.cpp`, `funcs.h` (not shown). Outline what should be in `funcs.cpp` and `funcs.h`; make sure you clearly explain the purpose of `#include "funcs.h"`.

[9 marks]

```

/*----- Tr5b.cpp -----
#include <iostream>
using namespace std;

// A
int stars(int n);
int spaces(int n);

int main(){
    int h = 5;  int nSp = 0; int nSt = 0;  int nNl = 0;
    for(int j= 0; j< h; j++){
        nSp += spaces(h - 1 - j);
        int m = j + 1;
        // cout<< "m = "<< m<< endl; // B1
        nSt += stars(m);
        nl(); ++nNl;    // line 23
        // cout<< "m = "<< m<< endl; // B2
    }
    int nc =  nSp + nSt + nNl;
    cout<< "Number of characters = "<< nc<< endl;
    return 0;
}

void nl(){
    cout<< '\n';
}
int stars(int n){
    int nc = 0;
    for(int i= 0; i< n; i++){
        cout<< '*'; nc++;
    }
    n = n + 10;
    return nc;
}
int spaces(int n){
    for(int i= 0; i< n; i++){
        cout<< ' ';
    }
    return n;
}
// Output
*
**
***
****
*****
Number of characters = 30

```

Figure 14: Tr5b.cpp

```

/*----- Tr5c.cpp -----
#include "funs.h" // E
using namespace std;

int main(){
    int h = 5;
    int nSp = 0;
    int nSt = 0;
    int nNl = 0;
    for(int j= 0; j< h; j++){
        nSp += spaces(h - 1 - j);
        nSt += stars(j + 1);
        nl(); ++nNl;
    }
    int nc = nSp + nSt + nNl;
    cout<< "Number of characters = "<< nc<< endl;
    return 0;
}

```

Figure 15: Tr5c.cpp

6. Write short notes on any **five** of the following.

- (a) `std::vector` versus plain C++ arrays.
- (b) C++ *templates*.
- (c) *Pass by value* versus *pass by reference*.
- (d) *Dynamic binding* and *virtual functions*.
- (e) `inline` functions.
- (f) *Inheritance* versus *inclusion* (or *composition*) in C++ classes.
- (g) *Deep copy* versus *shallow copy* in objects that contain pointers to heap memory.
- (h) *Destructors* in C++ classes.

Appendix A. Trigonometry.

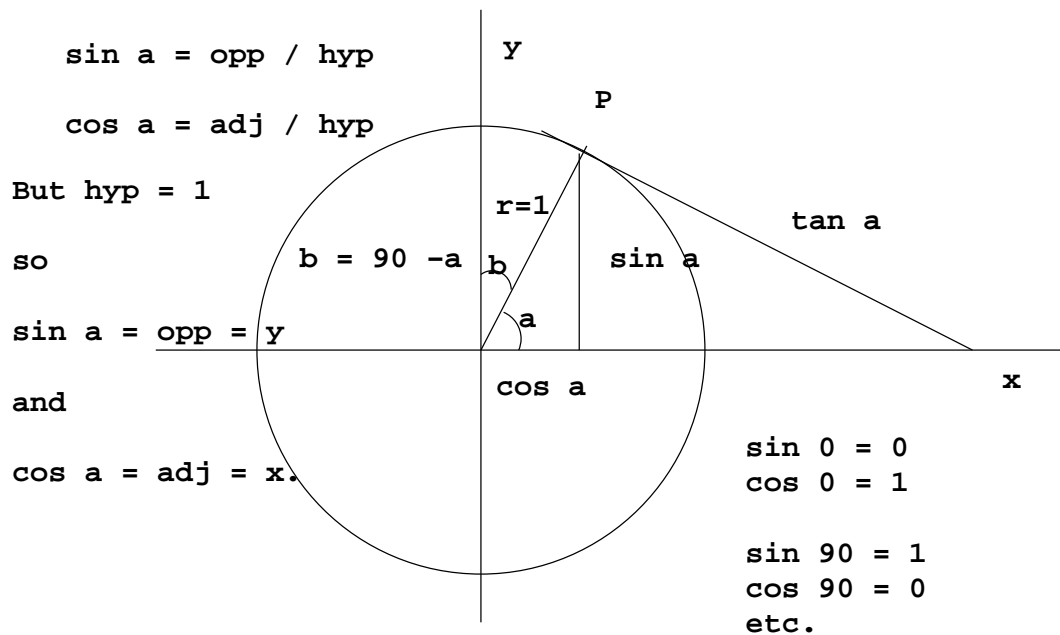


Figure 16: Sin, cos and circle.

From Figure 16 we can see from Pythagoras's Theorem that

$$\sin^2 \theta + \cos^2 \theta = 1. \quad (3)$$

Some values. $\sin 0^\circ = 0$, $\sin 30^\circ = 0.5$, $\sin 60^\circ = \sqrt{3}/2 = 0.866$, $\sin 90^\circ = 1$.
 $\cos 0^\circ = 1$, $\cos 30^\circ = \sqrt{3}/2 = 0.866$, $\cos 60^\circ = 0.5$, $\cos 90^\circ = 0$.

Radians.

A radian is about 57° ; it is the angle subtended by an arc of length r on a circle of radius r .

$\pi/2$ radians = 90° , π radians = 180° , 2π radians = 360° , etc.

$\pi = 3.141592635 \dots$

Degrees, d , to radians, r : $r = (180/\pi) \times d$.

Cos and Sin are periodic over 2π radians or 360° .

cos and sin repeat themselves after 2π radians or 360° .

Sin is an odd function: $\sin -\theta = -\sin \theta$.

Cos is an even function: $\cos -\theta = \cos \theta$.

Useful equations.

$$\sin(\theta + \phi) = \sin \theta \cos \phi + \cos \theta \sin \phi, \quad (4)$$

$$\sin(\theta - \phi) = \sin \theta \cos \phi - \cos \theta \sin \phi, \quad (5)$$

$$\cos(\theta + \phi) = \cos \theta \cos \phi - \sin \theta \sin \phi, \quad (6)$$

$$\cos(\theta - \phi) = \cos \theta \cos \phi + \sin \theta \sin \phi. \quad (7)$$

$$\sin^2 \theta + \cos^2 \theta = 1. \quad (8)$$

$$\sin^2 \theta = \frac{1}{2}(1 - \cos 2\theta). \quad (9)$$

$$\cos^2 \theta = \frac{1}{2}(1 + \cos 2\theta). \quad (10)$$

$$\cos 2\theta = \cos^2 \theta - \sin^2 \theta = 1 - \sin^2 \theta = 2 \cos^2 \theta - 1. \quad (11)$$

Appendix B. 2D Linear Transformations.

Scaling

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix}. \quad (12)$$

Rotation

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} \cos b & -\sin b \\ \sin b & \cos b \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix}. \quad (13)$$

Shear

 Shear along the x axis,

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix}. \quad (14)$$

Shear along the y axis,

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ b & 1 \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix}. \quad (15)$$

Reflection

 Reflect about the y axis (x-coordinates are negated),

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix}. \quad (16)$$

Reflect about the x axis (y-coordinates are negated),

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix}. \quad (17)$$

Projection

 Projection onto x-axis,

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix}. \quad (18)$$

Projection onto the y-axis,

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix}. \quad (19)$$

Translation

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} + \begin{bmatrix} u_x \\ u_y \end{bmatrix}. \quad (20)$$

Appendix C. 3D Affine Transformations using Homogeneous Coordinates.

Translation

$$\begin{bmatrix} v_x \\ v_y \\ v_z \\ v_w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_x \\ u_y \\ u_z \\ u_w \end{bmatrix}. \quad (21)$$

Rotation about the z-axis

$$R_z(b) = \begin{bmatrix} \cos b & -\sin b & 0 & 0 \\ \sin b & \cos b & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (22)$$

Rotation about the x-axis

$$R_x(b) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos b & -\sin b & 0 \\ 0 & \sin b & \cos b & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (23)$$

Rotation about the y-axis

$$R_y(b) = \begin{bmatrix} \cos b & 0 & \sin b & 0 \\ 0 & 1 & 0 & 0 \\ -\sin b & 0 & \cos b & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (24)$$

Appendix D. 3D Projection Transformations using Homogeneous Coordinates.

Perspective Transformation

$$\begin{bmatrix} -x'p_z \\ -y'p_z \\ -z'p_z \\ w' \end{bmatrix} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2nf}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}. \quad (25)$$

```
void glFrustum(GLdouble left, GLdouble right,
               GLdouble bottom, GLdouble top,
               GLdouble near, GLdouble far).
```

Orthographic Transformation

$$\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & -\frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (26)$$

```
void glOrtho(GLdouble left, GLdouble right,
              GLdouble bottom, GLdouble top,
              GLdouble near, GLdouble far).
```

Appendix E. OpenGL Graphics Primitives.

Figure 17 shows the list of possible graphics primitives and possible arguments for `glBegin` and Figure 18 gives a diagrammatic explanation.

<code>GL_POINTS</code>	individual points
<code>GL_LINES</code>	pairs of vertices interpreted as individual line segments
<code>GL_POLYGON</code>	boundary of a simple, convex polygon
<code>GL_TRIANGLES</code>	triples of vertices interpreted as triangles
<code>GL_QUADS</code>	quadruples of vertices interpreted as four-sided polygons
<code>GL_LINE_STRIP</code>	series of connected line segments
<code>GL_LINE_LOOP</code>	same as above, with a segment added between last and first vertices
<code>GL_TRIANGLE_STRIP</code>	linked strip of triangles
<code>GL_TRIANGLE_FAN</code>	linked fan of triangles
<code>GL_QUAD_STRIP</code>	linked strip of quadrilaterals

Figure 17: Geometric Primitive Names and Meanings.

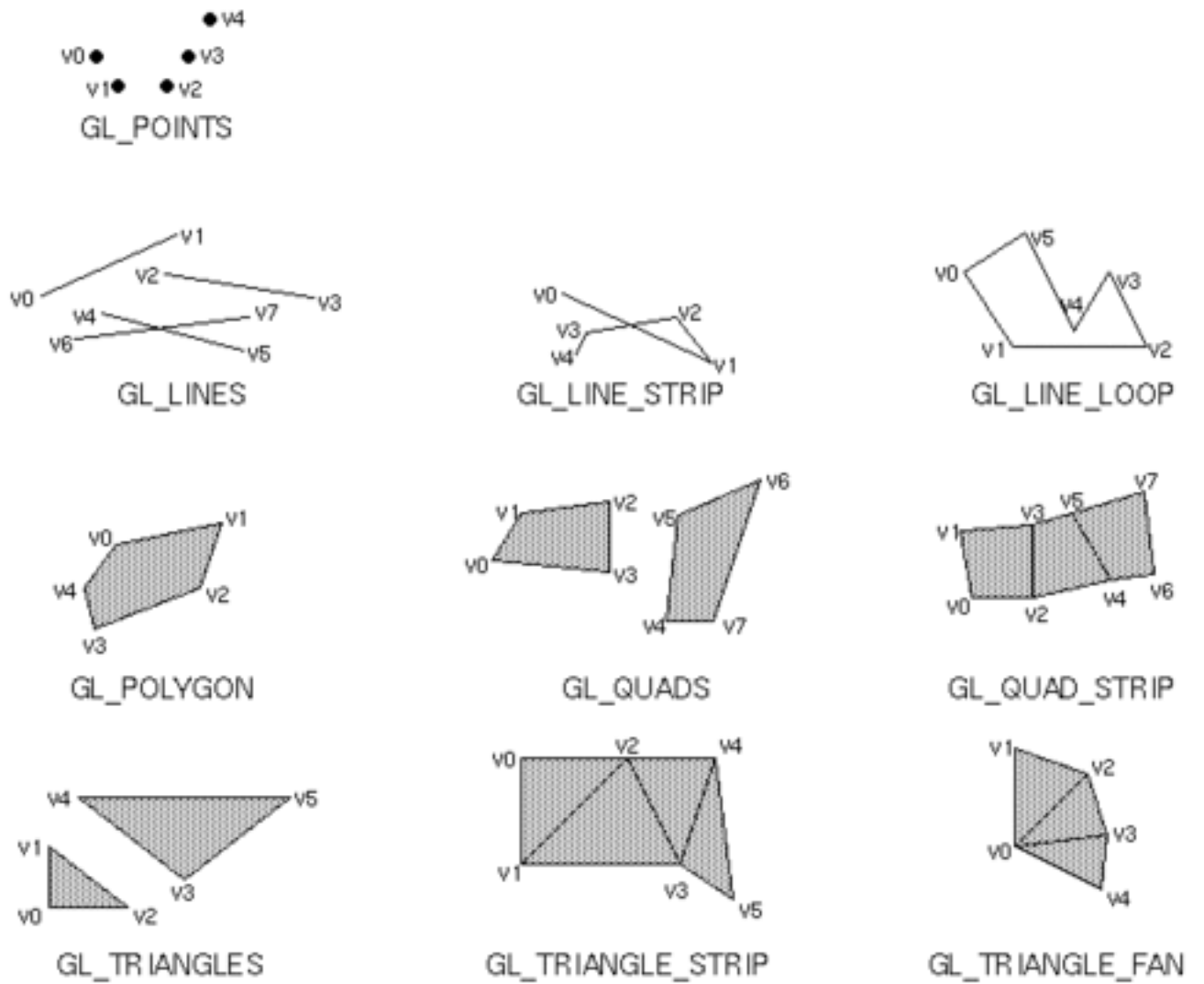


Figure 18: Geometric Primitive Types.