3D Graphics Programming Tools

Colours and Lighting

Dr. Xianhui Cherry Che

x.che@qmul.ac.uk

Learning Objectives

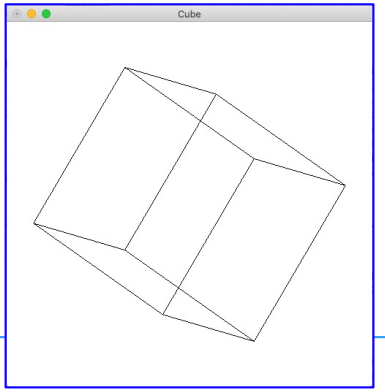
- Understand the importance of applying colours and lightings in OpenGL graphics especially in 3D objects
- Learn various techniques to apply lighting in OpenGL and what effects they can achive

Topics

- OpenGL Colours
- OpenGL Light

Recap: Draw a Cube – Full Program

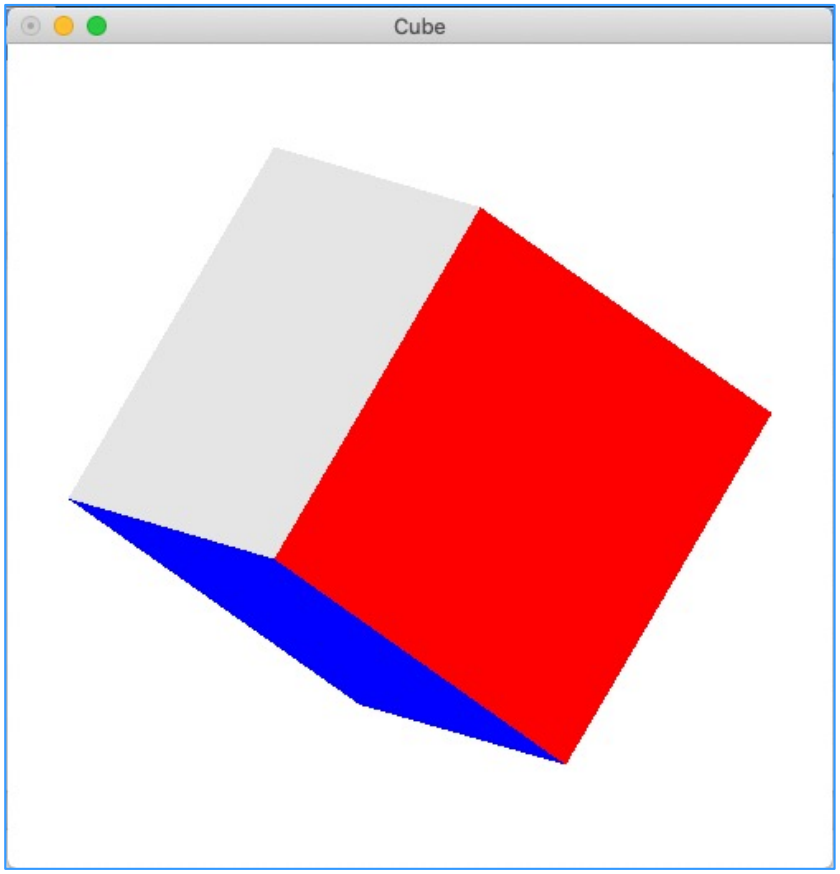
```
#include <GLUT/glut.h>
GLfloat X = 0.5;
static GLfloat theta[] = {45.0,45.0,45.0};
GLfloat CubeVertices[][3] = {{-1.0,-1.0,1.0},{-1.0,1.0,1.0}, {1.0,1.0,1.0}, {1.0,-1.0,1.0},
{-1.0,-1.0,-1.0}, {-1.0,1.0,-1.0}, {1.0,1.0,-1.0},{1.0,-1.0,-1.0}};

void a3dpolygon(GLfloat vertices[][3],int a,int b,int c,int d) {
    glBegin(GL_LINE_LOOP);
    glVertex3fv(vertices[a]);
    glVertex3fv(vertices[b]);
    glVertex3fv(vertices[c]);
    glVertex3fv(vertices[d]);
    glEnd();
}
void cube(){
    a3dpolygon(CubeVertices, 0,3,2,1);
    a3dpolygon(CubeVertices, 2,3,7,6);
    a3dpolygon(CubeVertices, 3,0,4,7);
    a3dpolygon(CubeVertices, 1,2,6,5);
    a3dpolygon(CubeVertices, 4,5,6,7);
    a3dpolygon(CubeVertices, 5,4,0,1);
}
void display(){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glRotatef(theta[0], 1.0, 0.0, 0.0);
    glRotatef(theta[1], 0.0, 1.0, 0.0);
    glRotatef(theta[2], 0.0, 0.0, 1.0);
    cube();
    glFlush();
}
```




```
void init(void) {
    int i, j;
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glColor3f (0.0, 0.0, 0.0);
    for (j = 0; j < 3; j++)
        for (i = 0; i < 8; i++)
        CubeVertices[i][j] = CubeVertices[i][j]*X;
}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB |
GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Cube");
    glutDisplayFunc(display);
    init();
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
}
```

Draw a Colour Cube

- Add colours to the cube



Draw a Colour Cube – Modification

```
#include <GLUT/glut.h>
GLfloat X = 0.5;
static GLfloat theta[] = {45.0,45.0,45.0};
GLfloat CubeVertices[][3] = {{-1.0,-1.0,1.0},{-1.0,1.0,1.0}, {1.0,1.0,1.0}, {1.0,-1.0,1.0},
{-1.0,-1.0,-1.0}, {-1.0,1.0,-1.0}, {1.0,1.0,-1.0},{1.0,-1.0,-1.0}};

void a3dpolygon(GLfloat vertices[][3],int a,int b,int c,int d) {
    glBegin(GL_LINE_LOOP);
    glVertex3fv(vertices[a]);
    glVertex3fv(vertices[b]);
    glVertex3fv(vertices[c]);
    glVertex3fv(vertices[d]);
    glEnd();
}
void cube(){
    a3dpolygon(CubeVertices, 0,3,2,1);
    a3dpolygon(CubeVertices, 2,3,7,6);
    a3dpolygon(CubeVertices, 3,0,4,7);
    a3dpolygon(CubeVertices, 1,2,6,5);
    a3dpolygon(CubeVertices, 4,5,6,7);
    a3dpolygon(CubeVertices, 5,4,0,1);
}
void display(){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glRotatef(theta[0], 1.0, 0.0, 0.0);
    glRotatef(theta[1], 0.0, 1.0, 0.0);
    glRotatef(theta[2], 0.0, 0.0, 1.0);
    cube();
    glFlush();
}
```

Declare an array of colour definitions here

Draw polygon instead for easier colour-filling

Set colour for each facet

```
void init(void) {
    int i, j;
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glColor3f (0.0, 0.0, 0.0);
    for (j = 0; j < 3; j++)
        for (i = 0; i < 8; i++)
        CubeVertices[i][j] = CubeVertices[i][j]*X;
}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB |
GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Cube");
    glutDisplayFunc(display);
    init();
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
}
```

Draw a Colour Cube – Modification

```
#include <GLUT/glut.h>
GLfloat X = 0.5;
static GLfloat theta[] = {45.0,45.0,45.0};
GLfloat CubeVertices[][3] = {{-1.0,-1.0,1.0},{-1.0,1.0,1.0}, {1.0,1.0,1.0}, {1.0,-1.0,1.0},
{-1.0,-1.0,-1.0}, {-1.0,1.0,-1.0}, {1.0,1.0,-1.0},{1.0,-1.0,-1.0}};
GLfloat colors[][3] = {{1.0,1.0,0.0},{0.0,1.0,0.0}, {1.0,0.0,0.0}, {1.0,0.5,0.0},
{0.9,0.9,0.9},{0.0,0.0,1.0}};

void a3dpolygon(GLfloat vertices[][3],int a,int b,int c,int d) {
    glBegin(GL_POLYGON);
    glVertex3fv(vertices[a]);
    glVertex3fv(vertices[b]);
    glVertex3fv(vertices[c]);
    glVertex3fv(vertices[d]);
    glEnd();}
void cube(){
    glColor3fv(colors[0]);
    a3dpolygon(CubeVertices, 0,3,2,1);
    glColor3fv(colors[1]);
    a3dpolygon(CubeVertices, 2,3,7,6);
    glColor3fv(colors[2]);
    a3dpolygon(CubeVertices, 3,0,4,7);
    glColor3fv(colors[3]);
    a3dpolygon(CubeVertices, 1,2,6,5);
    glColor3fv(colors[4]);
    a3dpolygon(CubeVertices, 4,5,6,7);
    glColor3fv(colors[5]);
    a3dpolygon(CubeVertices, 5,4,0,1);}
```

```
void display(){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glRotatef(theta[0], 1.0, 0.0, 0.0);
    glRotatef(theta[1], 0.0, 1.0, 0.0);
    glRotatef(theta[2], 0.0, 0.0, 1.0);
    cube();
    glFlush();}
void init(void) {
    int i, j;
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glColor3f (0.0, 0.0, 0.0);
    for (j = 0; j < 3; j++)
        for (i = 0; i < 8; i++)
        CubeVertices[i][j] = CubeVertices[i][j]*X;
}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Cube");
    glutDisplayFunc(display);
    init();
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
}
```

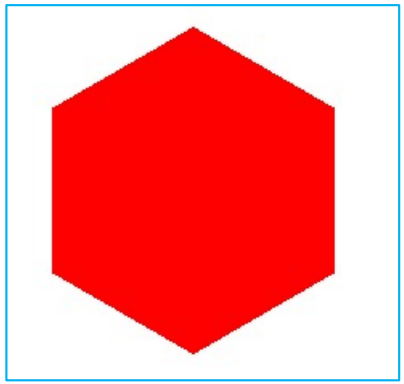
Unchanged

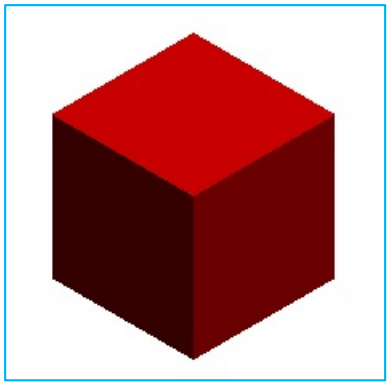
Topics

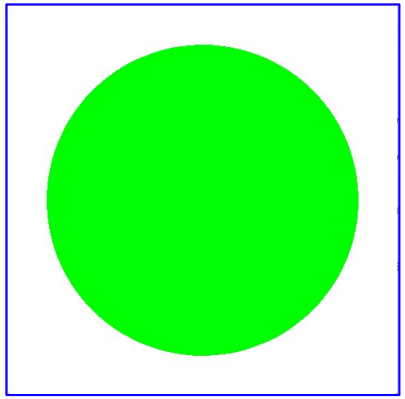
- OpenGL Colours
- OpenGL Light

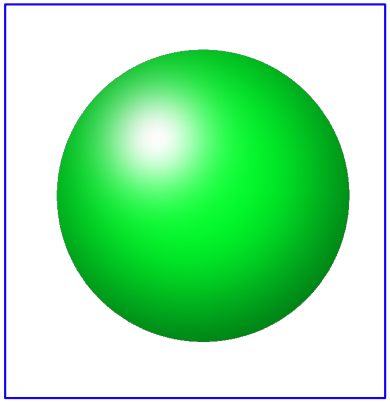
The Use of Light

- Adding colours to solid objects can be quite meaningless unless there is light exposure.







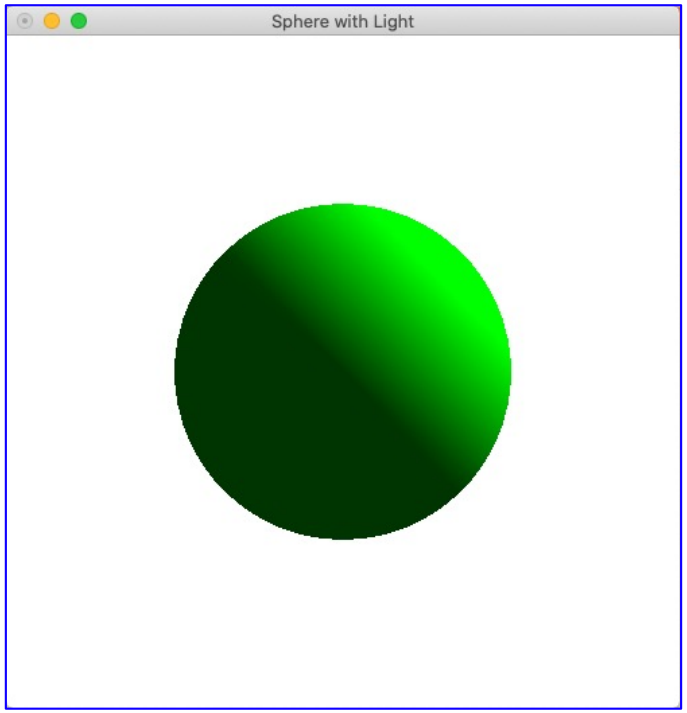


3D objects without Light

3D objects with Light

Basic OpenGL Lighting

- Three steps to add a basic lighting source:
  1. Enable lighting
  2. Define the position of the light source
  3. Set materials (colours)

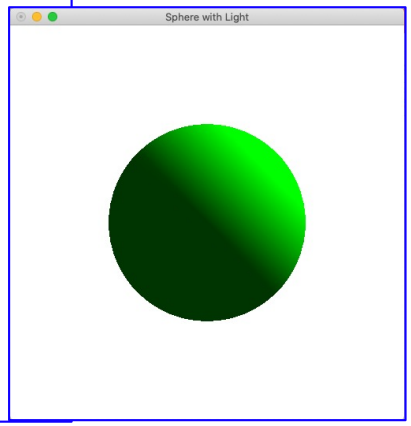


Sphere with Light

```
#include <GLUT/glut.h>

void init(void) {
    glClearColor(1.0, 1.0, 1.0, 1.0);

    //Step 1: Enable lighting
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);

    //Step 2: Set lighting position
    GLfloat light_position[] = { 1.0, 1.0, 0.0, 0.0 };
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);

    //Step 3: Set materials (colours)
    glEnable(GL_COLOR_MATERIAL);
   }
```

```
void display(){
    glClear(GL_COLOR_BUFFER_BIT |
GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glColor3f(0, 1, 0);
    glutSolidSphere(0.5, 50, 50);
    glFlush ();
}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE |
GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Sphere with Light");
    init();
    glutDisplayFunc(display);
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
}
```

Unchanged



Enable Lighting

- At least 8 light sources: 0 – 7
- Explicitly enable the lighting:

```
glEnable(GL_LIGHTING);
```

- To disable lighting:

```
glDisable(GL_LIGHTING) ;
```

- Each light source also needs to be explicitly enabled:

```
glEnable(GL_LIGHT0);
```

Set Lighting Parameters

- **`glLight`** — set light source parameters

```
void glLightf( GLenum light,
               GLenum pname,
               GLfloat param);

void glLighti( GLenum light,
               GLenum pname,
               GLint param);

void glLightfv( GLenum light,
                GLenum pname,
                const GLfloat * params);

void glLightiv( GLenum light,
                GLenum pname,
                const GLint * params);
```

Parameter Names:

GL_AMBIENT

GL_DIFFUSE

GL_SPECULAR

GL_POSITION

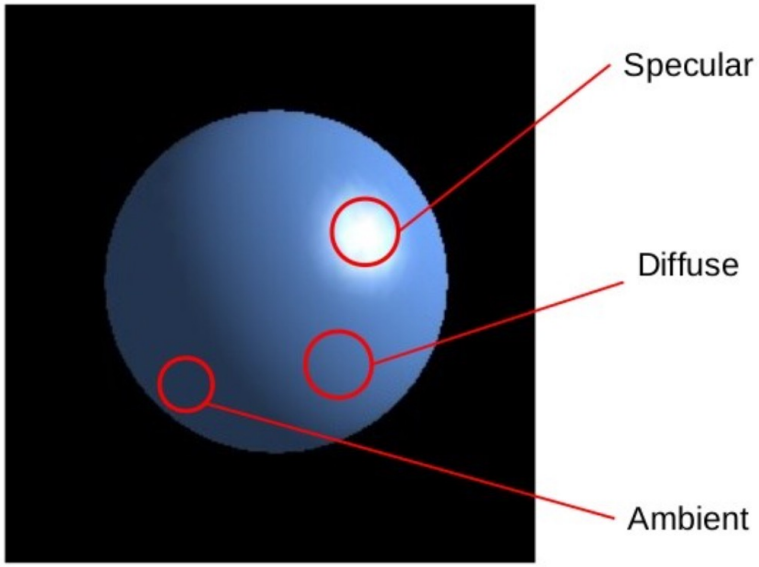
GL_SPOT_DIRECTION

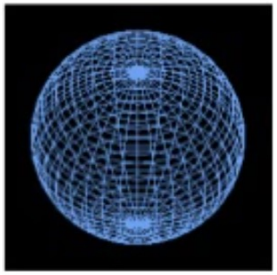
GL_SPOT_EXPONENT

GL_SPOT_CUTOFF

Different Types of Light

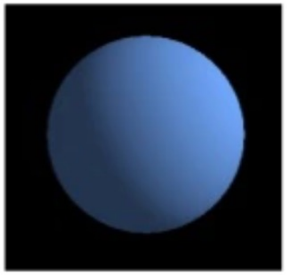
- **Specular:** sets of colours for highlight
- **Diffuse:** the colour of the object when it is illuminated
- **Ambient:** the colour of the mesh when it is not illuminated
- **Emissive:** the type of light which is being emitted by the object

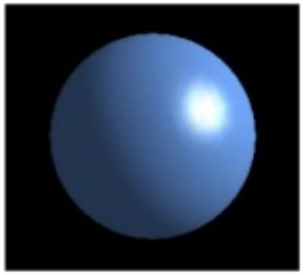


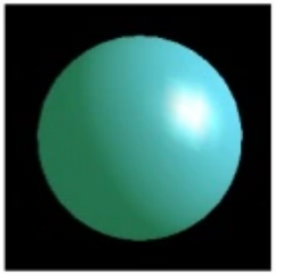

Wireframe


1. Original Color


2. Original Color + Ambient


3. Original Color + Ambient + Diffuse


4. Original Color + Ambient + Diffuse + Specular


5. Original Color + Ambient + Diffuse + Specular + Emissive

References

Parameter Name	Default Value	Meaning
GL_AMBIENT	(0.0, 0.0, 0.0, 1.0)	ambient RGBA intensity of light
GL_DIFFUSE	(1.0, 1.0, 1.0, 1.0)	diffuse RGBA intensity of light
GL_SPECULAR	(1.0, 1.0, 1.0, 1.0)	specular RGBA intensity of light
GL_POSITION	(0.0, 0.0, 1.0, 0.0)	(x, y, z, w) position of light
GL_SPOT_DIRECTION	(0.0, 0.0, -1.0)	(x, y, z) direction of spotlight
GL_SPOT_EXPONENT	0.0	spotlight exponent
GL_SPOT_CUTOFF	180.0	spotlight cutoff angle
GL_CONSTANT_ATTENUATION	1.0	constant attenuation factor
GL_LINEAR_ATTENUATION	0.0	linear attenuation factor
GL_QUADRATIC_ATTENUATION	0.0	quadratic attenuation factor

Specify Light Colours

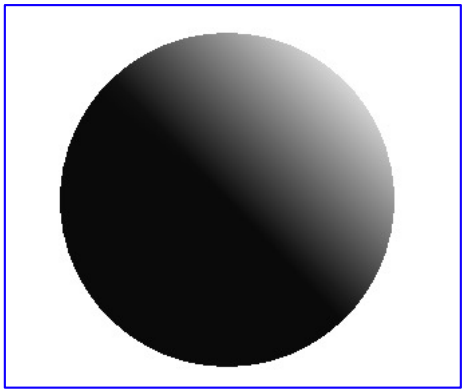
- If you do not want to use the default values, you can specify the colours using the **`glLight*`** function.
- For example:

```
glLightf(GL_LIGHT0, GL_AMBIENT, (0.15, 0.15, 0.15, 1.0));
glLightf(GL_LIGHT0, GL_DIFFUSE, (0.75, 0.75, 0.75, 1.0));
glLightf(GL_LIGHT0, GL_SPECULAR, (1, 1, 1, 1.0));
```

Material Colours

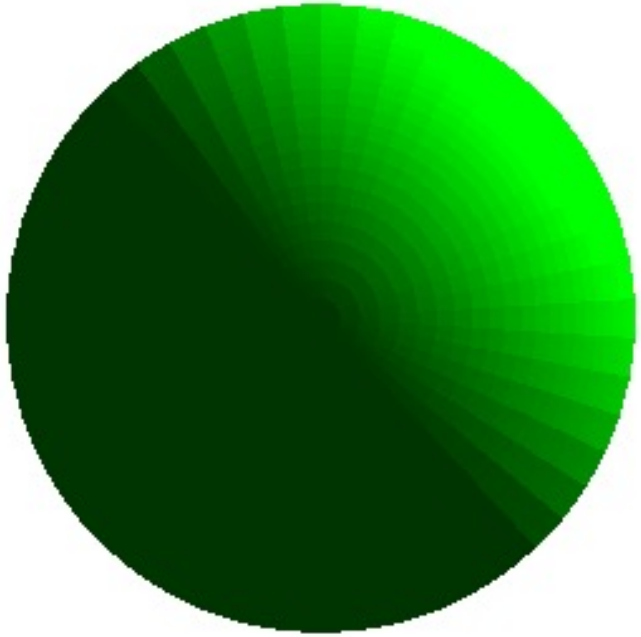
- When lighting is enabled, a vertex colour is not determined from the colour set by **`glColor`**, but by the currently set material colours combined with the light colours using the lighting computations.
- In order to set an object colour, you need to change the material setting (which by default is a diffuse grey material) before rendering, using the **`glMaterial`** functions.
- By calling **`glEnable(GL_COLOR_MATERIAL)`** and setting an appropriate mapping with **`glColorMaterial`** you can configure OpenGL to change a specific material colour, whenever you change the current vertex colour using either **`glColor`**.

Without enabling colour material, glColor is no longer being used

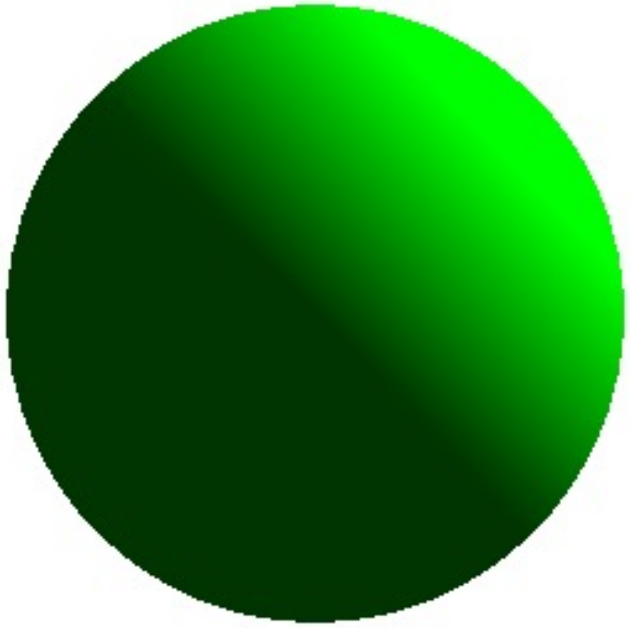


Shade Model

- `glShadeModel` function specifies the shading technique, which can be **`GL_FLAT`** and **`GL_SMOOTH`**. The default is **`GL_SMOOTH`**.



50 slices
50 stacks



`GL_Flat` **`GL_SMOOTH`**

Questions?

x.che@qmul.ac.uk