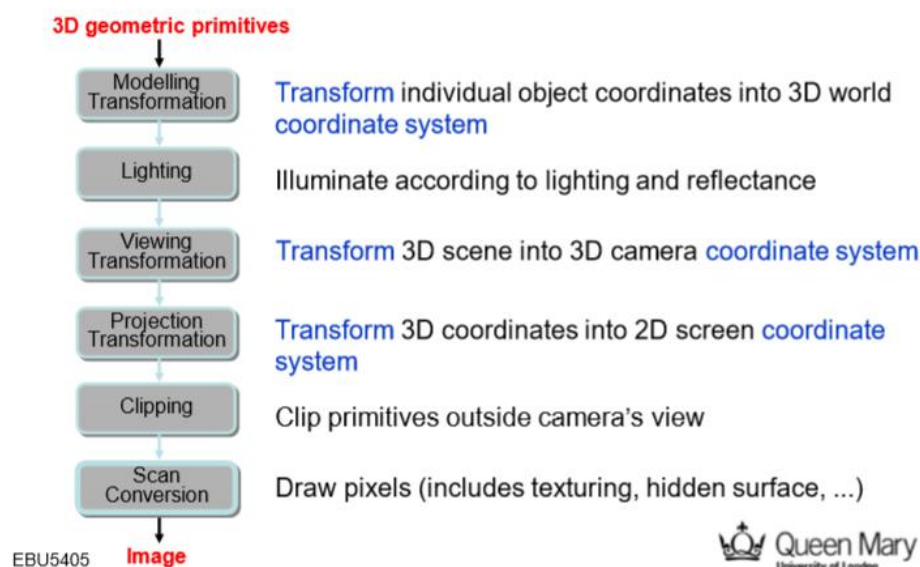
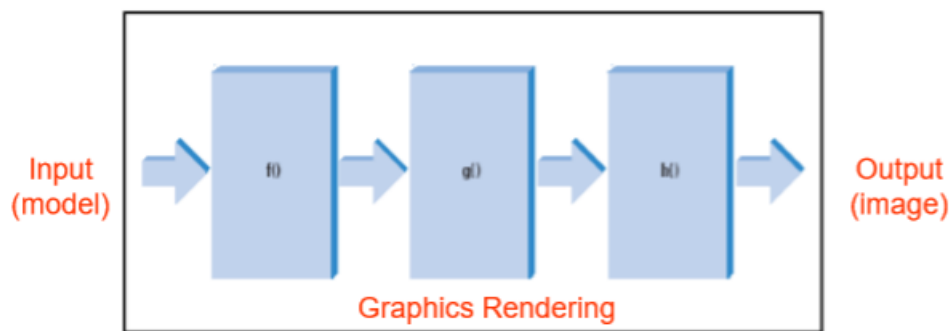


1. give my own definition of 3D Graphics programming
3D graphics programming is the subject concerned with how 2D images can be generated from abstract descriptions of 3D objects.
2. explain the terms: modelling, rendering and imaging
 - a) The generation of abstract descriptions of 3D objects is called geometric modelling.
 - b) The generation of 2D images from 3D models is called rendering.
The **3D objects of the 3D scene**, characterised by:
 - Their **geometry** - Their respective **positions** - Their **material** (used for shading calculations)
 A **camera** (a viewer), characterised by:
 - Its **position** - Its **lens** (used for projection calculations)
 A **light source**, characterised by:
 - Its **position** - Its **geometry** - Its **colour**
 - c) Imaging – Representing 2D images
3. explain the benefits of a pipeline architecture
 - 1) Graphics rendering is like a **manufacturing assembly** line with each stage adding something to the previous one.
 - 2) Within a pipeline architecture, all stages are working in **parallel**.
 - 3) Because of this pipeline architecture, today's graphics processing units (**GPUs**) **perform billions of calculations per second**. They are increasingly designed with more memory and more stages, so that more data can be worked on at the same time.
4. describe what each step of the 3D Graphics pipeline does

模型变换-光照-视角变换-投影变换-剪裁-扫描转换 1718



5. describe the natures of the input and output of the 3D Graphics pipeline



6. explain what a 3D Graphics transformation is
Modelling, viewing, projection
7. link the three OpenGL libraries and explain why OpenGL uses three libraries 1617
- OpenGL **core** library – OpenGL32 on **Windows**
 - OpenGL **Utility** Library (GLU) – Uses OpenGL core but **avoids having to rewrite** code
 - OpenGL **Utility** Toolkit (GLUT)
 - Links with **window system** (e.g. AGL for Macintosh)
 - Provides **functionality common to all window systems**
 - a) Open a window
 - b) Get input from mouse and keyboard
 - c) Menus
 - d) Event-driven
 - Code is portable but GLUT lacks the functionality of a good toolkit for a specific platform
8. compile and run a simple OpenGL program that opens a window on my display

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

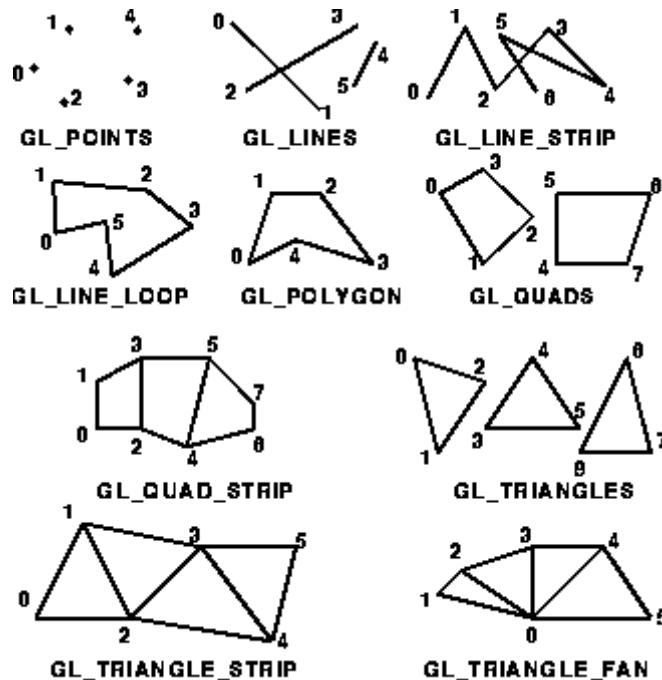
void display() {
    glClearColor(1.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutCreateWindow("Hello GLUT!");
    glutDisplayFunc(display);
    glutMainLoop();
}
```

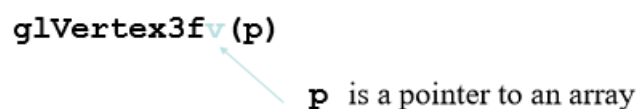
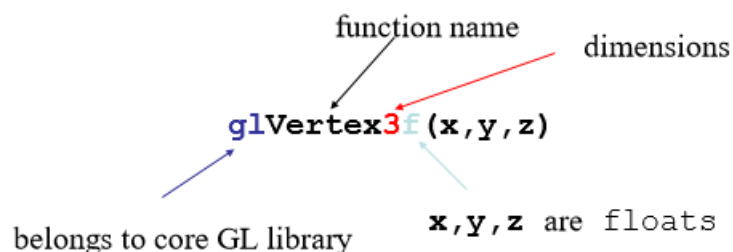
9. write a display callback function to change the window background colour
Note that the **main function** of the simple.c program defines a display callback function named **display**.
The display callback is executed whenever OpenGL decides the display **must be**

refreshed, for example when the window is opened.

10. use OpenGL primitives to make graphics appear inside the window



11. recognise, from the name of an OpenGL function: the library it belongs to, its purpose, and the number and type of parameters it requires



12. explain how OpenGL operates as a state machine

State changing • Transformation functions • Attribute functions

It means that once the value of a property is set, the value persists until a new value is set.

For example, if we use `glColor` command to set the current drawing colour to black, black will be used to draw ALL objects until we use `glColor` command again to

change the drawing colour.

13. use OpenGL functions to change the variables' default values for: colours, window parameters and viewing

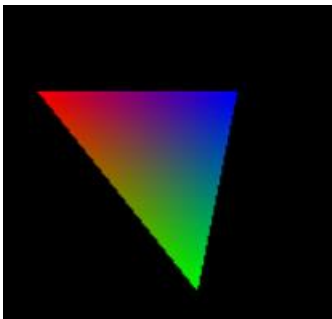
a) glColor3f the color values range from 0.0 (none) to 1.0 (all), whereas in glColor3ub the values range from 0 to 255

8 bits per component in frame buffer

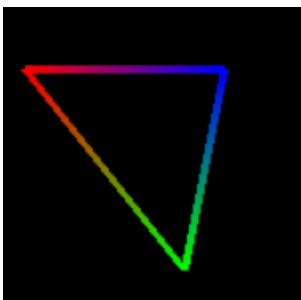
Colors and other attributes are not part of the object but are assigned when the object is rendered.

```
1 glBegin(GL_TRIANGLES);  
2     glColor3f(1.0f, 0.0f, 0.0f);  
3     glVertex3f(10.0f, 100.0f, 0.0f);  
4  
5     glColor3f(0.0f, 1.0f, 0.0f);  
6     glVertex3f(50.0f, 50.0f, 0.0f);  
7  
8     glColor3f(0.0f, 0.0f, 1.0f);  
9     glVertex3f(60.0f, 100.0f, 0.0f);  
10 glEnd();
```

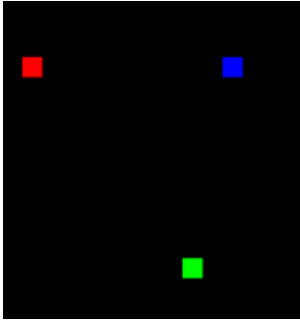
面默认的绘制方式为填充，即 glPolygonMode(GL_FRONT_AND_BACK, GL_FILL)



设置 glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)



glPolygonMode(GL_FRONT_AND_BACK, GL_POINT)



14. recognise if an OpenGL function is a “state-changing”, “primitive-generating” or “query” function **1718**

- OpenGL functions are of two types:

Primitive generating:

- a) Can cause output if primitive is **visible** (- Points - Line Segments - Polygons ...)
- b) How vertices are processed and the appearance of primitives are controlled by the state

State changing

- a) Transformation functions (- Modelling - Viewing)
- b) Attribute functions
 - 1) - Colour (points, lines, polygons)
 - 2) - Size and width (points, lines)
 - 3) - Stipple pattern (lines, polygons)
 - 4) - Polygon mode • Display as filled: solid color or stipple pattern • Display edges • Display vertices

Query function

- We can also use query functions

- `glGetIntegerv`

- `glGetFloatv`

to obtain any value that is part of the state

15. use the `glOrtho` function to change the size and position of the graphics inside the window **1718**

`glOrtho(left, right, bottom, top, near, far)`

假设有一个球体，半径为 1，圆心在(0, 0, 0)，那么，我们设定 `glOrtho(-1.5, 1.5, -1.5,`

`1.5, -10, 10)`;就表示用一个宽高都是 3 的框框把这个球体整个都装进了进来。

```

#include <GL/glut.h>  ← includes gl.h

int main(int argc, char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("simple");
    glutDisplayFunc(mydisplay);
    myinit();
    glutMainLoop();
}

void myinit()
{
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glColor3f(1.0, 1.0, 1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
}

```

Annotations for the code:

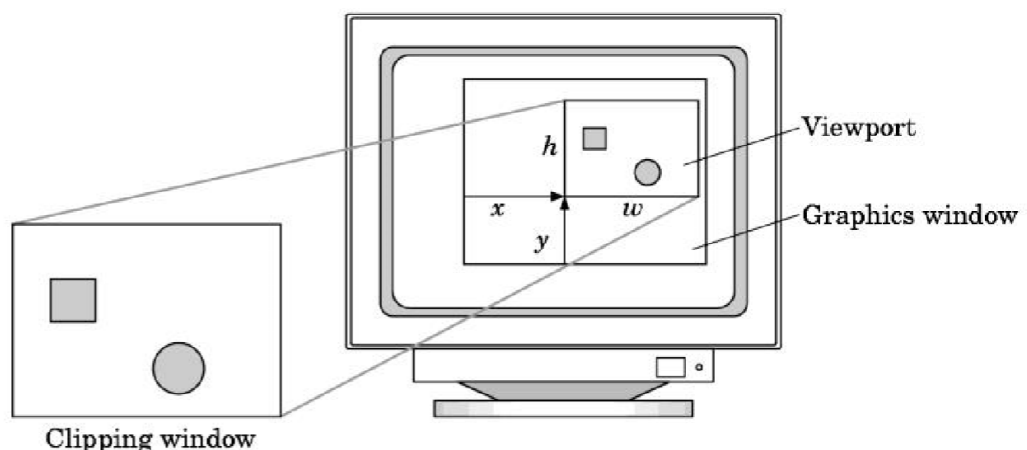
- `includes gl.h` points to `#include <GL/glut.h>`
- `define window properties` points to `glutInitDisplayMode`, `glutInitWindowSize`, and `glutInitWindowPosition`
- `display callback` points to `glutDisplayFunc(mydisplay)`
- `used to set OpenGL state` points to `myinit()`
- `enter event loop` points to `glutMainLoop()`
- `black clear color` points to `glClearColor(0.0, 0.0, 0.0, 1.0)`
- `opaque colour` points to `1.0` in `glClearColor`
- `fill/draw with white` points to `glColor3f(1.0, 1.0, 1.0)`
- `viewing volume` points to `glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0)`

16. use viewports

- We do not have to use the entire window for the image: **glViewport(x,y,w,h)**

Position of the window and size of the viewports

- Values in pixels (screen coordinates)



在已有窗口进行查看，即使视口大小与窗口大小不同，只要比例相同就不会形变，只是“未显示任何图像”的区域就多了。或者是说把他强行按照比例(w,h)

放置在某个位置(x,y)。

17. OpenGL Camera

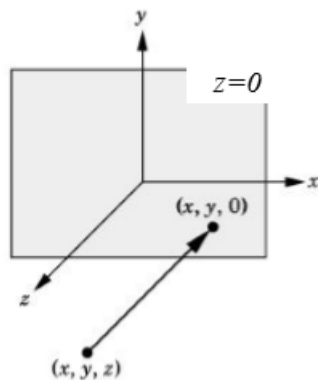
`glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);` 真正的截取

(left, right, bottom, top, near, far)

OpenGL places a default camera at the origin in object space pointing in the negative z direction and default z = 0.

The default viewing volume is a box centered at the origin with a side of length 2

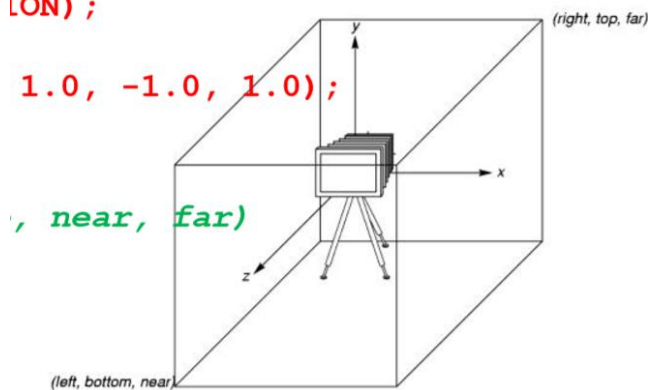
`gluOrtho2D(left, right, bottom, top)`



`[ON);`

`1.0, -1.0, 1.0);`

`, near, far)`



18. explain how OpenGL is event-driven

- Most systems have more than one input device, each of which can be triggered at an arbitrary time by a user
- Each trigger generates an event whose measure is put in an event queue which can be examined by the user program



- Window: resize, expose, iconify
- Mouse: click one or more buttons
- Motion: move mouse
- Keyboard: press or release a key
- Idle: nonevent - Define what should be done if no other event is in the queue

19. write callback functions to animate and interact with graphics using the mouse and the keyboard

```
glutMouseFunc(mymouse)
void mouse(int btn, int state, int x, int y) { if(btn==GLUT_RIGHT_BUTTON &&
state==GLUT_DOWN) exit(0); }
glutKeyboardFunc(mykey)
void mykey(unsigned char key, int x, int y)
{ if(key == 'Q' | key == 'q') exit(0); }
•Recall that the last line in main for a program using GLUT must be glutMainLoop();
which puts the program in an infinite event loop
```

```
    glutReshapeFunc (myreshape);
    glutMouseFunc (mymouse);
    glutDisplayFunc (mydisplay);
    glutMainLoop ();
}
```

- Every GLUT program must have a display callback

20. use double buffering

- Problem: the drawing of information in the frame buffer is decoupled from the display of its contents - Hence we can see partially drawn displays
- Instead of one color buffer, we use two - Front Buffer: one that is displayed but not written to - Back Buffer: one that is written to but not displayed
- Program must request a double buffer in main -glutInitDisplayMode(GL_RGB | GL_DOUBLE)
- At the end of the display callback, buffers are swapped

```
void mydisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);

    /* draw graphics here */

    glutSwapBuffers()
}
```


`-glutIdleFunc(myidle)`

- Useful for animations

```
void myidle() {  
    /* change something */  
    t += dt  
    glutPostRedisplay();  
}  
  
Void mydisplay() {  
    glClear();  
    /* draw something that depends on t */  
    glutSwapBuffers();  
}
```

21. write a reshape callback function to preserve the aspect ratio of my graphics

`glutReshapeFunc(myreshape)`

`void myreshape(int w, int h)`

- Returns width and height of new window (in pixels)
- A redisplay is posted automatically at the end of the execution of the callback
- Note: the reshape callback is a good place to put viewing functions because it is invoked when the window is first opened.
- This reshape preserves shapes by **making the viewport and world window** have the same aspect ratio.

```
void myReshape(int w, int h)  
{  
    glViewport(0, 0, w, h);  
    glMatrixMode(GL_PROJECTION); /* switch matrix mode */  
    glLoadIdentity();  
    if (w <= h)  
        gluOrtho2D(-2.0, 2.0, -2.0 * (GLfloat) h / (GLfloat) w,  
                    2.0 * (GLfloat) h / (GLfloat) w);  
    else gluOrtho2D(-2.0 * (GLfloat) w / (GLfloat) h, 2.0 *  
                    (GLfloat) w / (GLfloat) h, -2.0, 2.0);  
    glMatrixMode(GL_MODELVIEW); /* return to modelview mode */  
}
```

22. Posting redisplay

- Many events may invoke the display callback function - Can lead to multiple executions of the display callback on a single pass through the event loop
- We can avoid this problem by instead using `glutPostRedisplay()`; which sets a flag.
- GLUT checks to see if the flag is set at the end of the event loop
- If set then the display callback function is executed

23. use menus

Widgets: Most window systems provide a toolkit or library of functions for building user interfaces that use special types of windows called widgets.

Widget sets include tools such as

- Menus
- Slidebars

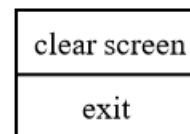
- Dials
- Input boxes
- GLUT supports pop-up menus - A menu can have submenus
- Three steps
- Define entries for the menu
- Define action for each menu item
- Attach menu to a mouse button

• In main.c

```
menu_id = glutCreateMenu(mymenu);
glutAddMenuEntry("clear Screen", 1);

glutAddMenuEntry("exit", 2);

glutAttachMenu(GLUT_RIGHT_BUTTON);
```



entries that appear when
right button depressed

identifiers

- Menu callback

```
void mymenu(int id)
{
    if(id == 1) glClear();
    if(id == 2) exit(0);
}
```

- Add submenus by glutAddSubMenu(char *submenu_name, submenu id)
24. obtain window size

```
glInt viewport[4];
glGetIntegerv(GL_VIEWPORT, viewport);
```

- viewport[0] = x
- viewport[1] = y
- viewport[2] = width
- viewport[3] = height

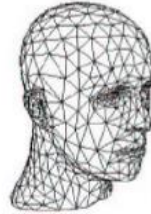
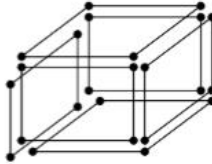
WEEK2

1. explain the differences between point-based, surface-based and constructive modelling

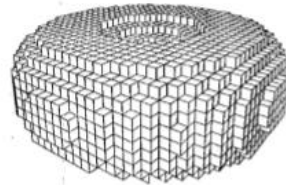
- Point-based



- Surface-based



- Constructive



EBU5405



2. give my own definition of a polygon mesh

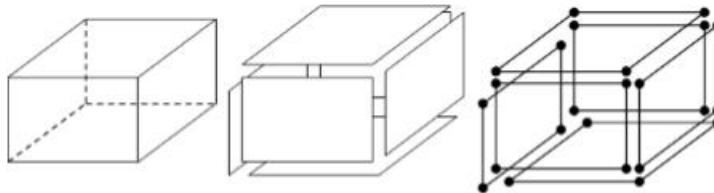
Polygon mesh (or "POLYHEDRON"):

- POLYHEDRON = geometric object with flat faces and straight edges
- Collection of polygons (faces), which are together connected to form the skin of the object **1718**

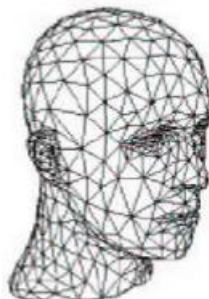
3. explain the differences between boundary representations and polygon meshes

Surface modelling

Boundary representations (b-reps)



Polygon mesh



EBU5405

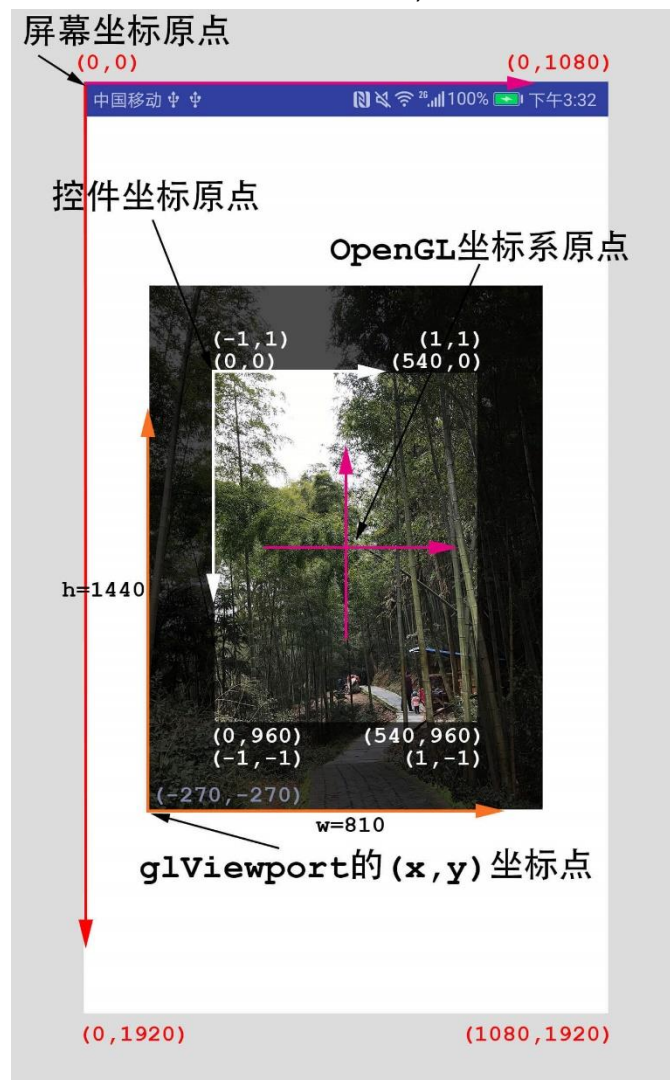


边界表示：将一幅图像分割成不同区域后，使用更适合于计算机进一步处理的形式，对得到的被分割的像素集进行表示和描述。

多边形网格：表示多面体形状的顶点与多边形的集合。

4. describe how polygon meshes are represented

- Faces Face list → indexes into vertices and normal lists, topological info
- Edges → the boundary between faces Edge list → indexes into end vertices of edges, topological info
- Vertices → the boundaries between edges, or where three or more faces meet. Vertex list → locations of the vertices, geometric info
- Normals, texture coordinates, colours, shading coefficients, etc Normal list → directions of the normal vectors, orientation info



5. explain why polygons are used for modelling

- They are: • Easy to represent • Easy to transform • Easy to draw

6. describe the properties polygons should have when used for modelling 1718

- Flat: all vertices are in the same plane
- Convex: All points on line segment between two points in a polygon are also in the polygon
- Simple: edges cannot cross

7. write a polygon mesh to model a simple geometric figure

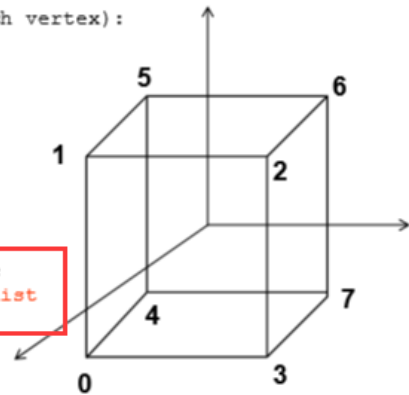
A cube:

Coordinates of **Vertices** (3D, x, y, and z for each vertex):

```
-1 , -1 , 1 # Vertex 0
-1 , 1 , 1 # Vertex 1
1 , 1 , 1 # Vertex 2
1 , -1 , 1 # Vertex 3
-1 , -1 , -1 # Vertex 4
-1 , 1 , -1 # Vertex 5
1 , 1 , -1 # Vertex 6
1 , -1 , -1 # Vertex 7
```

Lists of 6 **Faces** (vertices are referenced to the vertices above, -1 marks the end of the vertex list of a face) :

```
0, 3, 2, 1, -1 # Front face=vertex 0,1,2,3
2, 3, 7, 6, -1 # Right side
3, 0, 4, 7, -1 # Bottom
1, 2, 6, 5, -1 # Top
4, 5, 6, 7, -1 # Back
5, 4, 0, 1, -1 # Left
```



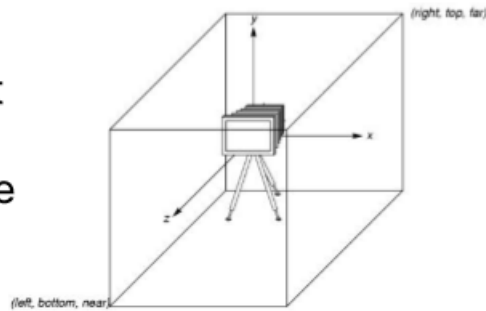
8. write programs that can draw graphics using polygon mesh models as inputs
9. compare the properties of some of the widely used 3D file formats: STL, OBJ, FBX, COLLADA, VRML and X3
 - STL encodes the surface geometry of a 3D model approximately using a triangular mesh.
 - The OBJ file format supports both approximate and precise encoding of surface geometry (i.e. smooth curves and surfaces such as NURBS: Non-Uniform Rational B-Spline), instead of polygons.
 - Used as an exchange format which facilitates high fidelity exchange between 3DS Max, Maya, MotionBuilder, Mudbox and other proprietary software.
 - The COLLADA format supports geometry, appearance related properties like color, material, textures, and animation. • The COLLADA format stores data using the XML markup language.
 - The VRML format uses a polygonal mesh to encode surface geometry and can store appearance related information such as color, texture, transparency etc • The X3D format adds NURBS encoding of the surface geometry, the capability of storing scene related information and support for animation

1. describe the default parameters of the OpenGL camera

OpenGL Camera

- OpenGL places a camera at the origin in object space pointing in the negative z direction

- Remember: the default viewing volume is a box centered at the origin with a side of length 2



2. use OpenGL functions to change the default parameters of the camera
`glMatrixMode (GL_PROJECTION);`
`glLoadIdentity();`
`glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);`
3. explain what hidden surface removal is
We want to see only those surfaces in front of other surface
4. describe how hidden surface removal is implemented in OpenGL
OpenGL uses a hidden-surface method called the **z-buffer algorithm** that **saves depth information as objects** are rendered so that **only the front objects appear in the image**
5. use OpenGL functions to request hidden surface removal

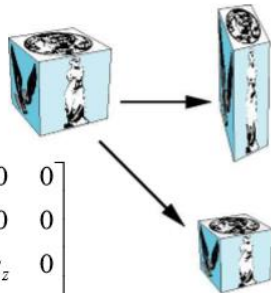
- It must be
 - Requested in `main.c`
 - `glutInitDisplayMode`
`(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH)`
 - Enabled in `init.c` or `main.c`
 - `glEnable(GL_DEPTH_TEST)`
 - Cleared in the display callback
 - `glClear(GL_COLOR_BUFFER_BIT |`
`GL_DEPTH_BUFFER_BIT)`

6. explain how hidden surface removal contributes to the visual perception of 3D shapes

`glEnable(GL_DEPTH_TEST)`, 启用了之后, OpenGL 在绘制的时候就会检查, 当前像素前面是否有别的像素, 如果别的像素挡道了它, 那它就不会绘制, 也就是说, OpenGL 就只绘制最前面的一层。

1. explain the difference between a model (or prototype or symbol) and an instance
 - In modelling, we often start with a simple object cantered at the origin, oriented with the axis, and at a standard size (a "model")
 - We apply an instance transformation to its vertices to
Scale Orient Locate
2. explain how modelling transformations result in a change of coordinate system
 - 1) Scaling
Expand or contract along each axis (fixed point of origin)
`glScaled(0.3, 0.3, 0.3);`

$$\begin{aligned}
 x' &= s_x x \\
 y' &= s_y y \\
 z' &= s_z z \\
 \mathbf{p}' &= \mathbf{S} \mathbf{p}
 \end{aligned}$$

$$\mathbf{S} = \mathbf{S}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$


The diagram illustrates scaling transformations. A central cube is shown with two arrows pointing to two other cubes. One arrow points to a smaller cube, representing contraction. The other arrow points to a taller, narrower rectangular prism, representing expansion along the y-axis and contraction along the x-axis.

Queen Mary
University of London

2) Rotation

$$\mathbf{R} = \mathbf{R}_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

`glRotated(θ , 0.0, 0.0, 1.0);`

- For rotation about x axis, x is unchanged

- For rotation about y axis, y is unchanged

$$\mathbf{R} = \mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{glRotated}(\theta, 1.0, 0.0, 0.0);$$

$$\mathbf{R} = \mathbf{R}_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{glRotated}(\theta, 0.0, 1.0, 0.0);$$

3) Translation

Move (translate, displace) a point to a new location

$$T = T(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{glTranslated}(d_x, d_y, d_z);$$

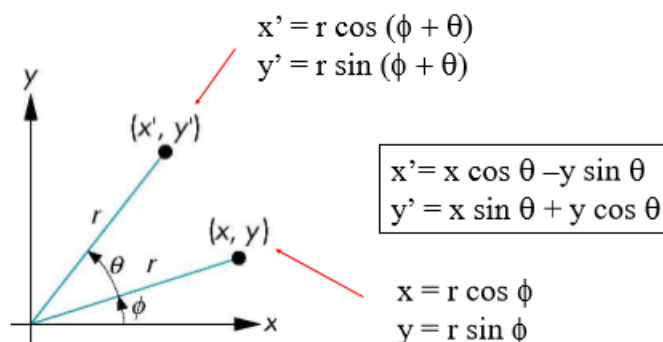
Rotation About a Fixed Point other than the Origin (Use Translation First)

3. explain why matrices are used to represent transformations
 - In OpenGL matrices are part of the state
 - Model-View (GL_MODELVIEW)
 - Projection (GL_PROJECTION)
 - Texture (GL_TEXTURE) (ignore for now) -
 - Color(GL_COLOR) (ignore for now)
 - Single set of functions for manipulation
 - Select which to manipulate with:
 - glMatrixMode(GL_MODELVIEW); -glMatrixMode(GL_PROJECTION);
4. write the general form of a 2D rotation around the origin transformation matrix

Rotation (2D)

Consider rotation about the origin by θ degrees

- radius stays the same, angle increases by θ



5. explain why homogenous coordinates are used
 - or in homogeneous coordinates

$$p' = R_z(\theta)p$$

A homogeneous coordinate system adds one value w to any point or vector, for example, 3D point will be defined with 4 values in homogeneous coordinates. They are used to allow the representation of translation transforms in matrix form, because translations are not linear transformations.

6. compare the properties of linear and **affine transformations**

This form is better for implementation because all affine transformations can be expressed this way and multiple transformations can be concatenated together.

NOTE: 仿射变换等价于一个线性变换后跟一个平移。

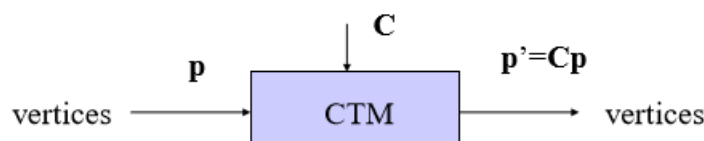
[An affine transformation is equivalent to a linear transformation followed by a translation.]

7. explain how matrix composition by post multiplication makes transformations efficient

Note that the **last operation specified is the first executed in the program.**

8. explain the role of the OpenGL “Current Transformation Matrix” (CTM)

Conceptually there is a 4×4 homogeneous coordinate matrix, the current transformation matrix (CTM) that is part of the state and is applied to all vertices that pass down the pipeline (recall that they are created by `glVertex`)



9. Using the OpenGL matrices **1718**

- In OpenGL the model-view matrix is used to
 - Position the camera which can be done by rotations and translations but it is often easier to use `gluLookAt`
 - Build models of objects.
- The projection matrix is used to define the view volume and to select a camera lens.

`GL_PROJECTION`, `GL_MODELVIEW` 和 `GL_TEXTURE`

```
glMatrixMode(GL_MODELVIEW);
```

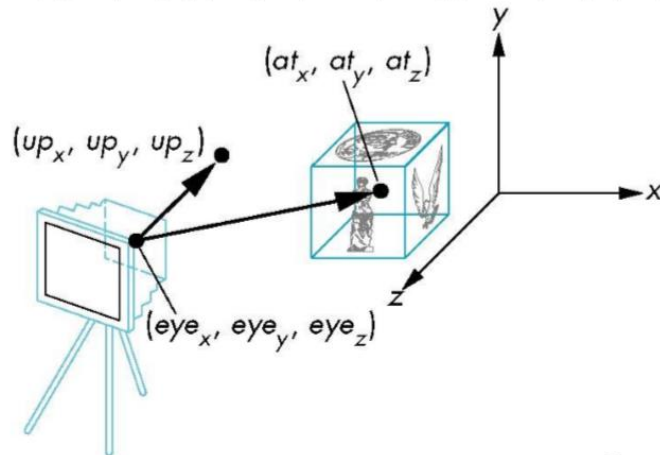
```
glLoadIdentity();
```

如果参数是 `GL_PROJECTION`，这个是投影的意思，就是要对投影相关进行操作，也就是把物体投影到一个平面上，就像我们照相一样，把 3 维物体投到 2 维的平面上。这样，接下来的语句可以是跟透视相关的函数，比如 `glFrustum()` 或 `gluPerspective()`; 如果参数是 `GL_MODELVIEW`，这个是对模型视景的操作，接下来的语句描绘一个以模型为基础的场景，这样来设置参数，

接下来用到的就是像 `gluLookAt()` 这样的函数；若是 `GL_TEXTURE`，就是对纹理相关进行操作；

gluLookAt

```
gluLookAt(eyex, eyey, eyez, atx, aty, atz, upx, upy, upz)
```



该函数定义一个视图矩阵，并与当前矩阵相乘。

第一组 `eyex, eyey, eyez` 相机在世界坐标的位置

第二组 `centerx, centery, centerz` 相机镜头对准的物体在世界坐标的位置

第三组 `upx, upy, upz` 相机向上的方向在世界坐标中的方向

你把相机想象成为你自己的脑袋：

第一组数据就是脑袋的位置

第二组数据就是眼睛看的物体的位置

第三组就是头顶朝向的方向（因为你可以歪着头看同一个物体）。

10. Matrix Stacks

In many situations we want to save transformation matrices for later use - E.g. when traversing hierarchical data structures

`glPushMatrix()`

`glPopMatrix()`

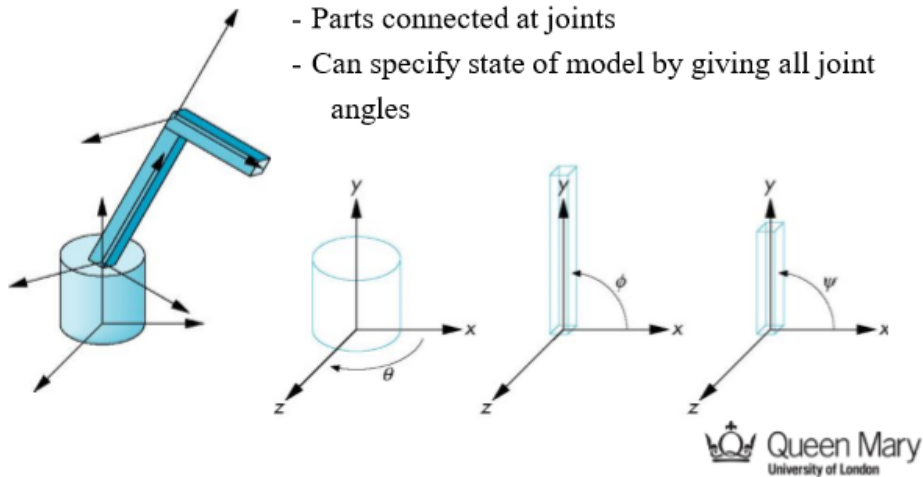
当经过一系列的变换后，栈顶矩阵被修改，此时调用 `glPopMatrix()` 时，栈顶矩阵被弹出，且又会恢复为原来的状态。

11. The OpenGL Camera

- In OpenGL, initially the object and camera frames are the same
- Default model-view matrix is an identity
- The camera is located at origin and points in the negative z direction
- OpenGL also specifies a default view volume that is a cube with sides of length 2 centered at the origin
- Default projection matrix is an identity

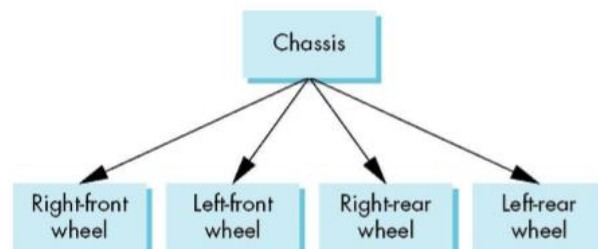
1. write the hierarchical model of a simple articulated 3D object
 1112 the transformation of one part is propagated to the related part.

- The robot arm is an example of an **articulated model**



42

Tree Model of Car



Allows to define the transform of the child with respect to the transform of the parent

7

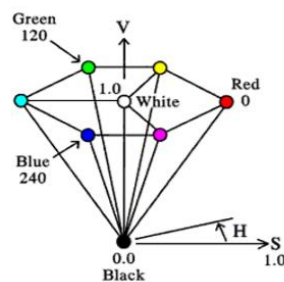
2. explain stack-based traversal
 Display of the tree requires a **graph traversal** 1718
 - **Visit each node once**
 - **Display function at each node** that describes the part associated with the node, applying the correct transformation matrix for position and orientation

week 3

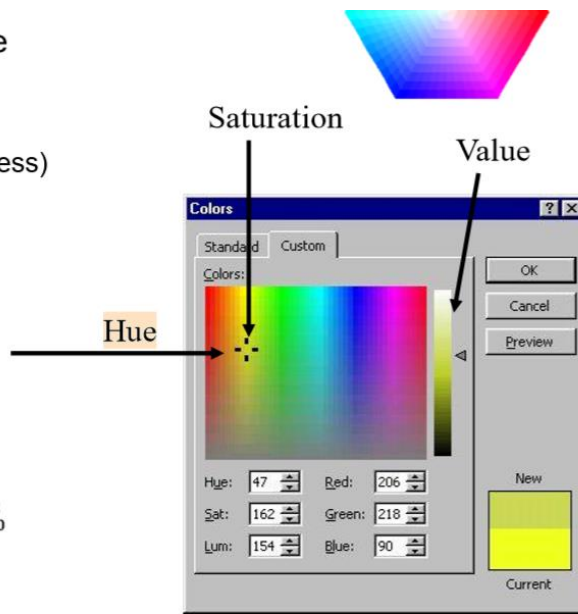
1. understand the terms hue, saturation and brightness
 - Hue → Distinguishes between colours like red, green, blue, etc.
 - Saturation → How far the color is from a gray of equal intensity
 - Lightness → The perceived intensity of a reflecting object
2. describe a colour (hue, saturation and brightness) expressed in the RGB model
3. describe a colour (hue, saturation and brightness) expressed in the HSV model

- Intuitive colour space

- H = Hue
- S = Saturation
- V = Value (or brightness)



EBU5405



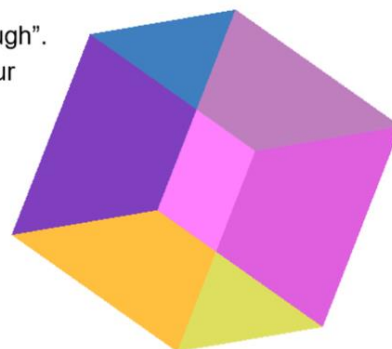
4. use OpenGL functions to change the drawing colour

- In RGBA mode, we use a fourth colour component: A or alpha, which is an **opacity**.
- An opacity of 1.0 means the colour is opaque and cannot be "seen through".
- An opacity of 0.0 means that a colour is transparent.

```
GLfloat colors[][4] = {{0.5,0.5,0.5,0.5},
{0.0,1.0,0.0,0.5}, {1.0,0.0,1.0,0.5},
{1.0,0.0,0.0,0.5}, {0.0,0.0,1.0,0.5},
{1.0,1.0,0.0,0.5}};

glColor4fv(colors[0]);
```

EBU5405



5. use OpenGL functions to reset the colour buffer


```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

6. use OpenGL functions to change the shading model

glShadeModel (GL_SMOOTH) or GL_FLAT

smooth: OpenGL interpolates vertex colours across visible polygons [default] 在可

见多边形之间插入顶点颜色

flat: Colour of first vertex determines fill colour 第一个顶点的颜色决定填充颜色

7. use OpenGL functions to change colours' transparency and blending properties
colour must be declared with degree of transparency using alpha value, e.g.

glColor4f(0.0, 0.0, 1.0, 0.3);

glEnable(GL_BLEND);

glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

Blend 混合是将源色和目标色以某种方式混合生成特效的技术。混合常用来绘制透明或半透明的物体。在混合中起关键作用的 α 值实际上是将源色和目标色按给定比率进行混合，以达到不同程度的透明。 α 值为 0 则完全透明， α 值为 1 则完全不透明。混合操作只能在 RGBA 模式下进行，颜色索引模式下无法指定 α 值。物体的绘制顺序会影响到 OpenGL 的混合处理。

8. understand the terms illumination, lighting and shading

- Illumination – the transport of energy from light sources to surfaces & points

- Lighting – the process of computing the luminous intensity

- Shading – the process of assigning colours to pixels

9. explain the difference between empirical and physical-based lighting models

- Illumination models → two categories:

- Empirical • **simple formulations** that approximate observed phenomenon 经验的主义的

- Physically based • models based on the **actual physics of light** interacting with matter

10. explain how lighting contributes to the visual perception of 3D shapes

11. explain the difference between direct and indirect illumination 1718

- 1) Ambient light sources: Simulate Indirect illumination from emitters, bouncing off intermediate surfaces

模拟来自发射器的间接照明，从媒介表面反弹

Use ambient light source 1415

- **No** spatial or directional characteristics (don't come from any direction:1)

- Illuminates all surfaces **equally**
- Amount reflected **depends on surface properties**

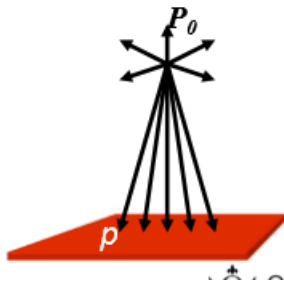
2)

a) Distant light sources

- **direction is constant** for all surfaces in the scene
- all rays of light from the source are **parallel**

b) Point light sources

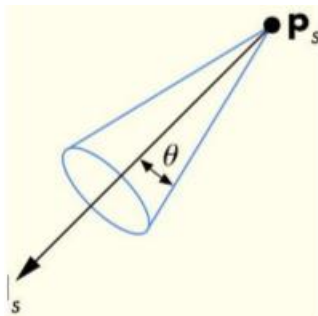
- **emits light equally in all directions** from a single point
- **the direction to the light from a point on a surface differs** for different points



c) Spotlights

- Characterised by a **narrow range of angles** through which light is emitted
- Realistic spotlights are characterised by the **distribution of light within the cone**(圆

锥)



12. 1112 describe the properties of diffuse reflection

Ideal diffuse reflection

- 1) is a very **rough** surface at the microscopic level 微观粗糙
- 2) an incoming ray of light is equally likely to be reflected in any direction over the hemisphere 入射光在半球的任何方向上都有相等可能被反射
- 3) The amount of light reflected depends on angle of incident light 反射光的数量

取决于入射光的角度 • follow Lambert's cosine law

13. Lambert's cosine law

- The energy reflected by a small portion of a surface from a light source

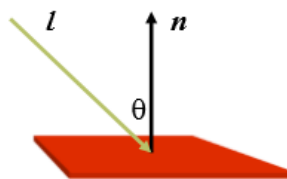
in a given direction is proportional to the cosine of the angle between that direction and the surface normal 一个光源在给定方向上反射的一小部分表面的能量与该方向与表面法线夹角的余弦成正比

- These are often called Lambertian surfaces
- Note that the reflected intensity is independent of the viewing direction, but does depend on the surface orientation with regard to the light source

反射强度与观察方向无关，但与光源的表面方向有关

14. compute diffuse reflection

- The angle between the surface normal and the incoming light is the *angle of incidence*:



$$I_{diffuse} = k_d I_{light} \cos \theta$$

In practice we use vector arithmetic:

$$I_{diffuse} = k_d I_{light} (n \cdot l)$$

EBU5405



15. describe the properties of **specular** reflection

Shiny surfaces exhibit specular reflection **in one direction only**

- A light shining on a specular surface **causes a bright spot** (specular highlight)
- Where these highlights appear is **a function of the viewer's position** → specular reflectance is **view dependent**

16. describe the Phong model's components

- Has three components –Diffuse –Specular –Ambient
- Uses four vectors –To source (l) –To viewer (v) –Normal (n) –Perfect reflector (r)

17. use OpenGL functions to enable lighting calculations

- Shading calculations are enabled by – **glEnable(GL_LIGHTING)** –Once lighting is enabled, **glColor() is ignored!!**
- Must **enable each light source individually** – **glEnable(GL_LIGHTi)** **i=0,1.....**
- Can choose light model **parameters** to **control the shading calculations** – **glLightModeli(parameter, GL_TRUE)**
 - **GL_LIGHT_MODEL_LOCAL_VIEWER** do not use simplifying distant

viewer assumption in calculation

- GL_LIGHT_MODEL_TWO_SIDED shades both sides of polygons independently

- glLightModel(GL_LIGHT_MODEL_AMBIENT, global_ambient); - To create **a small amount of ambient light** even when all the sources are turned off.

- GLfloat global_ambient[] = {1.0, 1.0, 1.0, 1.0};

18. use OpenGL functions to set up various types of light sources

1) Defining a Point Light Source:

- The position is given in homogeneous coordinates

If w = 1.0, we are specifying a finite location

If w = 0.0, we are specifying a parallel source with the given direction

```
GLfloat diffuse0[]={1.0, 0.0, 0.0, 1.0};
GLfloat ambient0[]={1.0, 0.0, 0.0, 1.0};
GLfloat specular0[]={1.0, 0.0, 0.0, 1.0};
GLfloat light0_pos[]={1.0, 2.0, 3.0, 1.0};
```

```
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glLightfv(GL_LIGHT0, GL_POSITION, light0_pos);
glLightfv(GL_LIGHT0, GL_AMBIENT, ambient0);
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse0);
glLightfv(GL_LIGHT0, GL_SPECULAR, specular0);
```

EBU5405



2) Spotlights

–Direction GL_SPOT_DIRECTION

–Angle GL_SPOT_CUTOFF

–Attenuation GL_SPOT_EXPONENT

- glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, direction);
- glLightfv(GL_LIGHT0, GL_SPOT_CUTOFF, cutoff);

3) Ambient Light

A red light in a white room will cause a red ambient term that disappears when the light is turned off

– glLightModel(GL_LIGHT_MODEL_AMBIENT, global_ambient)

19. use OpenGL functions to set up various material properties

glMaterialv()

- Material properties are also part of the OpenGL state
- Set by `glMaterialv()`

```
GLfloat ambient[] = {0.2, 0.2, 0.2, 1.0};
GLfloat diffuse[] = {0.8, 0.8, 0.8, 1.0};
GLfloat specular[] = {0.0, 0.0, 0.0, 1.0};
GLfloat shine[] = {100.0};
glMaterialf(GL_FRONT, GL_AMBIENT, ambient);
glMaterialf(GL_FRONT, GL_DIFFUSE, diffuse);
glMaterialf(GL_FRONT, GL_SPECULAR, specular);
glMaterialf(GL_FRONT, GL_SHININESS, shine);
```

20. Normals¹¹¹²

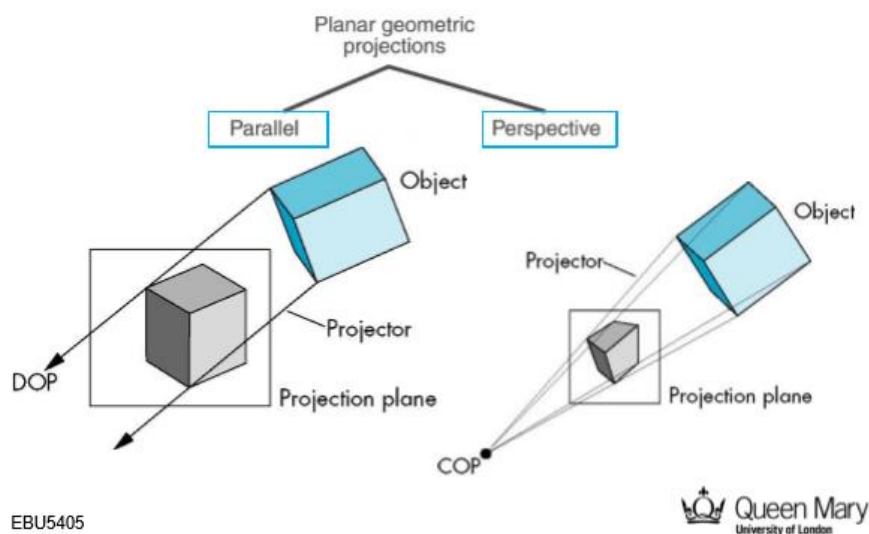
- Set by `glNormal*()` –`glNormal3f(x, y, z)`; –`glNormal3fv(p)`

unit length USE `glEnable(GL_NORMALIZE)` to normalize automatically.

defines the orientation of surface

used for calculating the amount of the light reflected by diffuse reflection.

1. explain how projection transformations result in a change of coordinate system
2. describe the properties of parallel projection

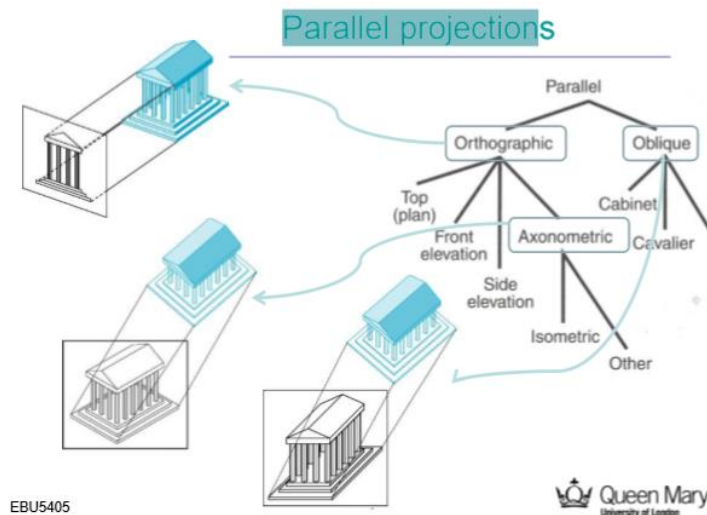


Centre of projection is at infinity 投影中心在无穷远处

– Direction of projection (DOP) is the same for all points 投影方向

DOP is perpendicular to the view plane

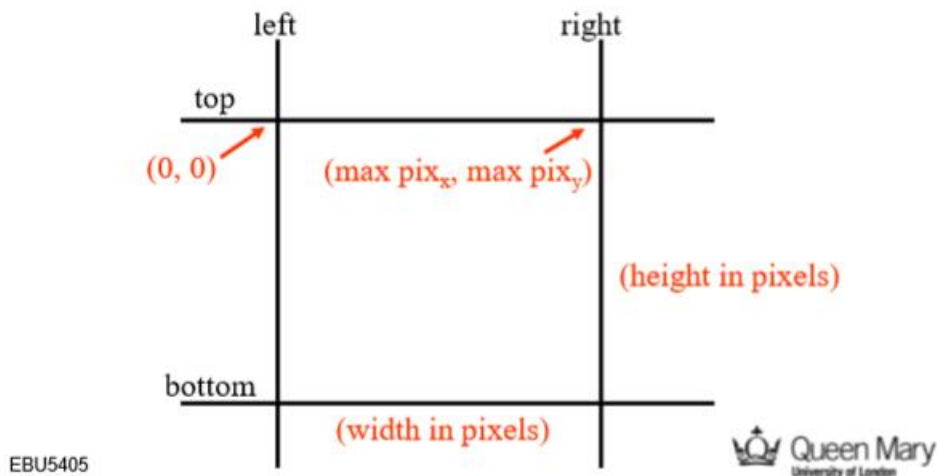
3. describe the different types of parallel projection



正投影，斜投影

4. write the matrix of a simple orthographic projection transformation PPT
5. explain the screen-space transformation matrix

- glOrtho (left, right, bottom, top, near, far)

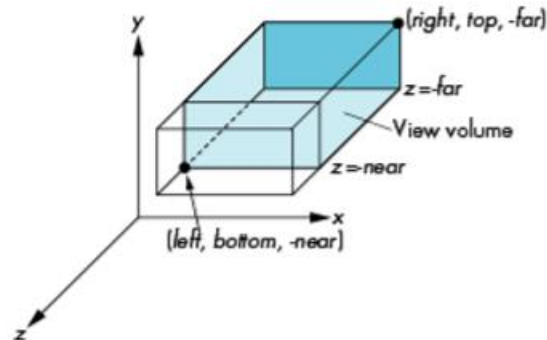


$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{\text{width}}{\text{right} - \text{left}} & 0 & 0 & \frac{-\text{left} \times \text{width}}{\text{right} - \text{left}} \\ 0 & \frac{\text{height}}{\text{bottom} - \text{top}} & 0 & \frac{-\text{top} \times \text{height}}{\text{bottom} - \text{top}} \\ 0 & 0 & \frac{z_{\text{max}}}{\text{far} - \text{near}} & \frac{-\text{near} \times z_{\text{max}}}{\text{far} - \text{near}} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- This matrix scales and translates to accomplish the transition in units
 - Left, right, top, bottom refer to the viewing frustum (view volume) in modelling coordinates glOrtho2D()
 - width and height are in pixel units (viewport)
- 6. describe the properties of perspective projection
 - objects exhibit perspective foreshortening (近大远小)
 - Parallel lines appear to converge to single point (vanishing point)
- 7. explain what is foreshortening (透视缩短)
 - distant objects appear smaller – objects closer to viewer look large
- 8. write the matrix of a perspective projection
 - PPT
- 9. use OpenGL functions to set the parameters of orthographic projection

OpenGL Orthogonal Viewing

`glOrtho(left, right, bottom, top, near, far)`



near and **far** measured from camera

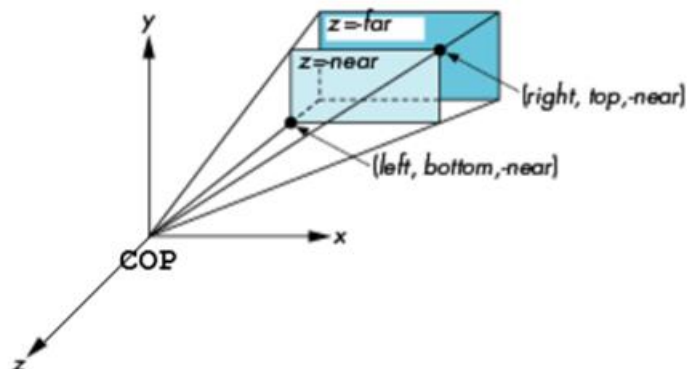
EBU5405



10. use OpenGL functions to set the parameters of perspective projection

OpenGL Perspective

`glFrustum(left, right, bottom, top, near, far)`

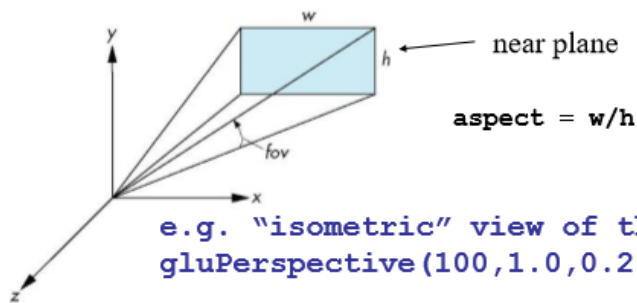


EBU5405



Using Field of View

- With **glFrustum** it is often difficult to get the desired view
- **gluPerspective(fov, aspect, near, far)** often provides a better interface



EBU5405

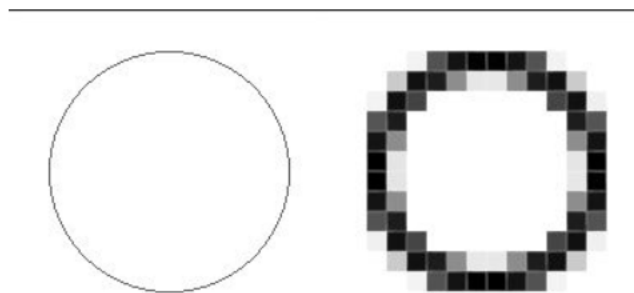
WEEK4

1. explain what rasterisation is

首先，光栅化 (Rasterize/rasterization)。

这个词儿Adobe官方翻译成栅格化或者像素化。没错，就是把矢量图形转化成像素点儿的过程。我们屏幕上显示的画面都是由像素组成，而三维物体都是点线面构成的。要让点线面，变成能在屏幕上显示的像素，就需要Rasterize这个过程。就是从矢量的点线面的描述，变成像素的描述。

如下图，这是一个放大了1200%的屏幕，前面是告诉计算机我有一个圆形，后面就是计算机把圆形转换成可以显示的像素点。这个过程就是Rasterize。



2. explain how triangulation can be done

A Delaunay triangulation for a set P of points in the plane is a triangulation $DT(P)$ such that no point in P is inside the circumcircle of any triangle in $DT(P)$.

3. recognise a good triangulation result from a bad one

4. explain the edge-walking rasterisation optimisation technique

- draw edges • interpolate colors down edges
- fill in horizontal spans for each scanline • at each scanline, interpolate edge colors across span

5. explain why convex polygons are easier to rasterise than concave polygons

- Convex polygons → for every pair of points in the polygon, the segment

between them is fully contained in the polygon 每一对点全都在在多边

- Concave polygons → Not convex: some two points in the polygon are joined by a segment not fully contained in the polygon

6. discuss triangle rasterisation issues

Sliver; Moving slivers ; Moving slivers

7. explain the edge-equations rasterisation optimisation technique

1213 Edge walking: After drawing the edges of triangles and interpolating colours down the edges, it consists in using scan lines to interpolate edge colours along these lines. 画完边界插完值之后，用扫描线确定边缘的颜色。

- Order the three triangle vertices in x and y

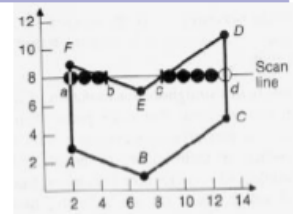
- Find middle point in y dimension and compute if it is to the left or right of polygon
- 8. discuss the use of a parity test for the rasterisation of random polygons

General polygon rasterisation

Basic idea: use a **parity test**

```

for each scanline
    edgeCnt = 0;
    for each pixel on scanline left to right
        if (oldpixel->newpixel crosses edge)
            edgeCnt ++;
        // draw the pixel if edgeCnt odd
        if (edgeCnt % 2)
            setPixel(pixel) ;
    
```



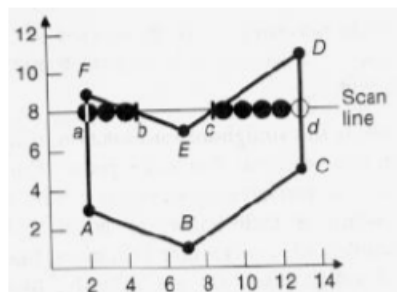
EBU5405



- 9. discuss the use of an active edge table for the rasterisation of random polygons

Active edge table

- Idea
 - Edges intersecting a given scanline are likely to intersect the **next scanline**
 - The order of edge intersections does not change much from scanline to scanline



EBU5405



The active-edge table is a data structure that consists of all the intersection points of the edges with the current scanline.

These intersection points are sorted by increasing x coordinate.

This allows the intersection points to be paired off, and be used for filling the scanline appropriately.

As the scan conversion moves on to the next scanline, the AET is updated so that it properly represents that scanline.

Active edge table

- Algorithm: for scanline from bottom to top...
 - Sort all edges by their **minimum y coordinate**
 - Starting at bottom, add edges with $Y_{\min} = 0$ to AET
 - For each scanline:
 - Sort edges in AET by **x intersection**
 - Walk from left to right, setting pixels by parity rule
 - Increment scanline
 - Retire edges with $Y_{\max} < Y$
 - Add edges with $Y_{\min} < Y$
 - Recalculate edge intersections
 - Stop when $Y > Y_{\max}$ for last edges

`gluLookAt()`是观察变换，`glOrtho()`是正交投影。
`gluLookAt()`是摄像机的位置，`glOrtho()`是将当前的可视空间设置为正投影空间。
`gluLookAt()`作用 MODELVIEW 矩阵，`glOrtho()`作用 PROJECTION 矩阵。

可有时画的图形看不见，有时又可以看见，那它们到底有什么联系呢？我们往下看：

先看看这两个函数的定义：

```
void glOrtho(GLdouble left,  
             GLdouble right,  
             GLdouble bottom,  
             GLdouble top,  
             GLdouble near,  
             GLdouble far);  
  
void gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez,  
              GLdouble centerx, GLdouble centery, GLdouble centerz  
,  
              GLdouble upx, GLdouble upy, GLdouble upz);
```

The rendering pipeline

- Move models • Illuminate • Move camera • Project to display • Clip • Rasterise

Modelling transformation:

`Gltranslated()` `glrotatef()` `glscaled()`

Viewing transformation:

`Glulookat()`9

Projection transformation

`glOtho()`6

`glInitDisplayMode()`

GLUT_RGB color model for drawing

GLUT_DOUBLE double buffering for writing and reading for animations

GLUT_DEPTH extra depth buffer to enable hidden surface removal

`glenable()`

GL_BLEND controls blending of RGBA values, i.e. how colors can mix together and form a new color. A transparency not turned on will be ignored

GL_DEPTH_TEST enables the depth buffer which stores a distance with the view plane for each pixel to hidden surface removal.

GL_LIGHTING enables lighting calculations as opposed to default flat shading. It leads to all `glColor` will be ignored.

Default:

Defaults

- Note that this program makes heavy use of variable default values for:
 - Colours (background and foreground)
 - Window parameters (e.g. size)
 - Viewing (camera parameters, e.g. where the square appears in the window ...)

```
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB)
glutShadeModel(GL_SMOOTH)
glcolor3f(0.0,0.0,0.0)
```

3 种缩小方式

`glScalef()`全减成一半

`glOrtho()`全放大两倍

`glutSolidCube()` 减小两倍

注意先 `glMatrixMode(GL_MODELVIEW)`和 `glLoadIdentity()`

Reflection: specular; diffuse

`glMaterialfv(GL_FRONT, GL_某个值)`

`GL_AMBIENT` how the material will reflect the ambient light it receives. RGB 的比
例

`GL_SHININESS` how close the surface is to a mirror for specular reflection. 多近

于表面可以导致镜面反射 5 low shininess 100 high shininess

`GL_EMISSION` turns the surface into a light source by addition emission term.

Rasteration:

An image described with geometric primitives converts into a raster image with pixels. Last step for graphics pipeline.

Z dimension is to compute for comparing the depth of several polygons and only draw the one that is closer to the projection plane. 深度 画离投影幕最近的多边形