ENSF 545
Introduction to Virtual Reality

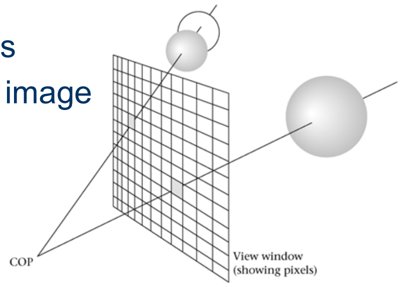
Advanced VR Programming

Camera, Projection, and View

Dr. Y. Hu

Camera (Eye)

- Simple camera is limiting and it is necessary to model a camera that can be moved.
- Parameters of a camera
  - Location (x, y, z)
  - Direction the camera points
  - Direction to be “up” on the image

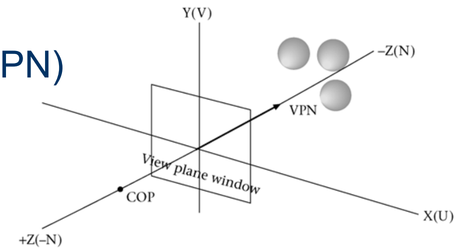


COP = centre of projection = view point

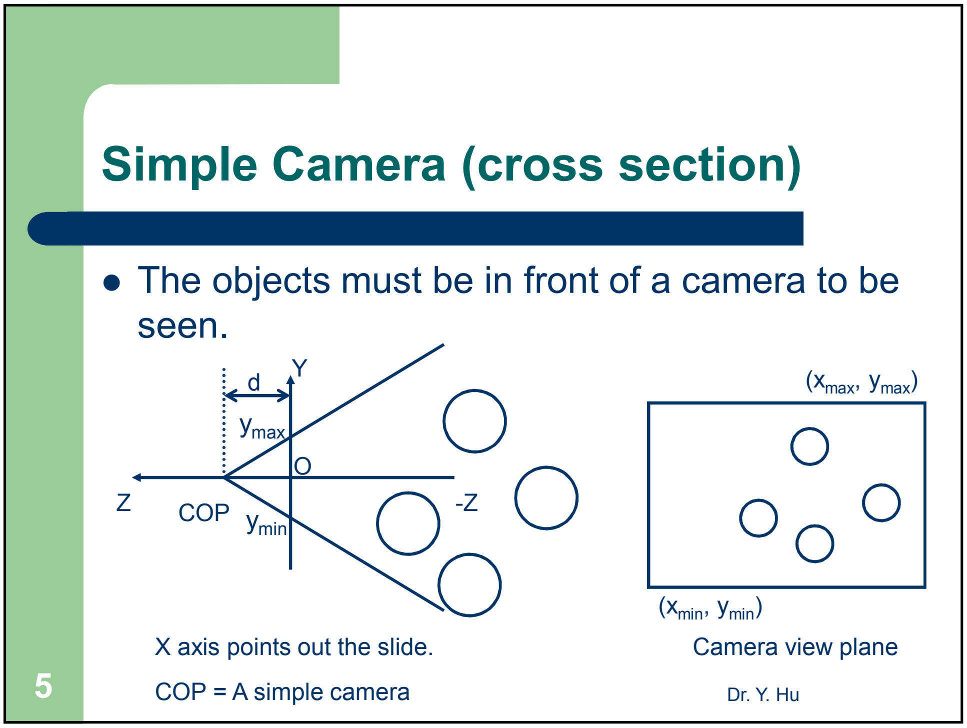
3

Camera Coordinate System

- Coordinate systems
  - Word (**XYZ**) – Right-handed
  - Camera (**UVN**) – Left-handed
- View reference point (VRP)
  - camera location
- View plane normal (VPN)
  - camera points to
- View up vector (VUV)
  - Camera up direction



4



Simple Camera (cross section)

- The objects must be in front of a camera to be seen.

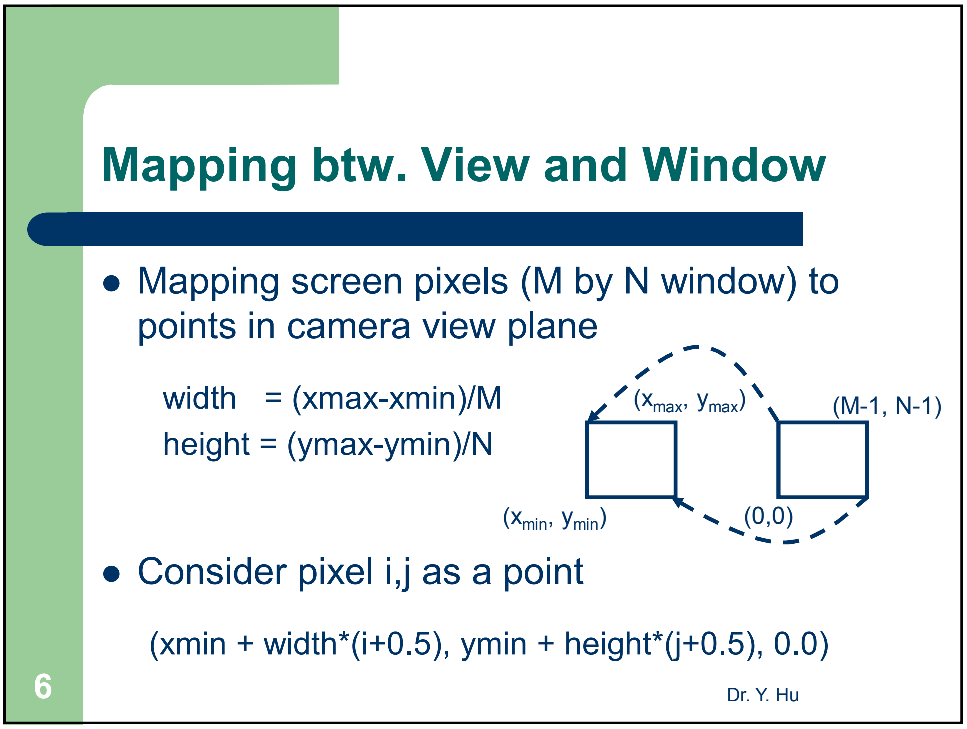
X axis points out the slide.

Camera view plane

COP = A simple camera

Dr. Y. Hu

5



Mapping btw. View and Window

- Mapping screen pixels (M by N window) to points in camera view plane

width = (xmax-xmin)/M
height = (ymax-ymin)/N

- Consider pixel i,j as a point

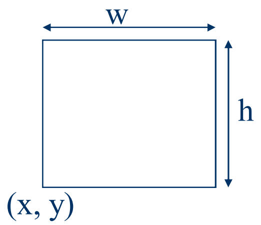
(xmin + width*(i+0.5), ymin + height*(j+0.5), 0.0)

Dr. Y. Hu

6

Defining a Viewport

- Clipping window ← **world coordinates**.
- OpenGL rendering ← **screen coordinates**.
- The drawing region on a screen → **`ViewPort`.**

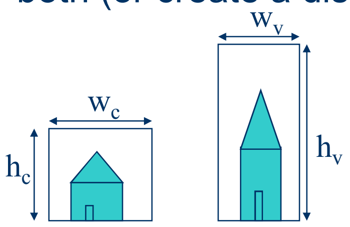


```
void glViewPort(GLint x, GLint y,
                GLsizei w, GLsizei h);
```

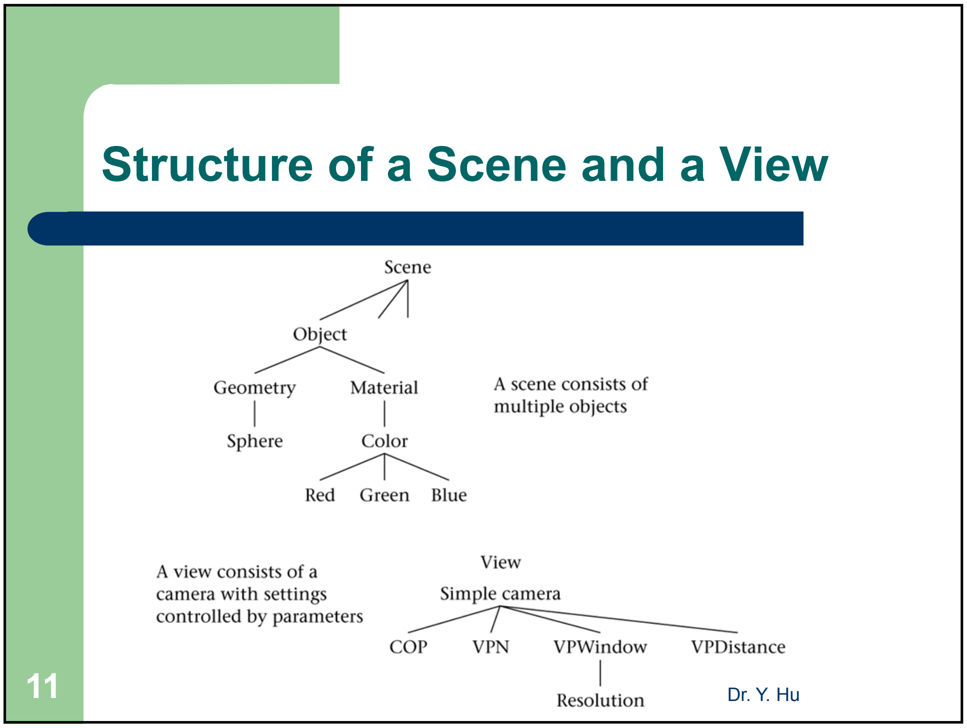
Lower left-hand corner

Mapping from World to Screen

- The clipping region ←→ The entire viewport.
- Make the height/width (aspect ratio) the same for both (or create a distorted image).



Clipping Viewport



Structure of a Scene and a View

Scene

Object

Geometry

Material

Sphere

Color

Red Green Blue

A scene consists of multiple objects

View

Simple camera

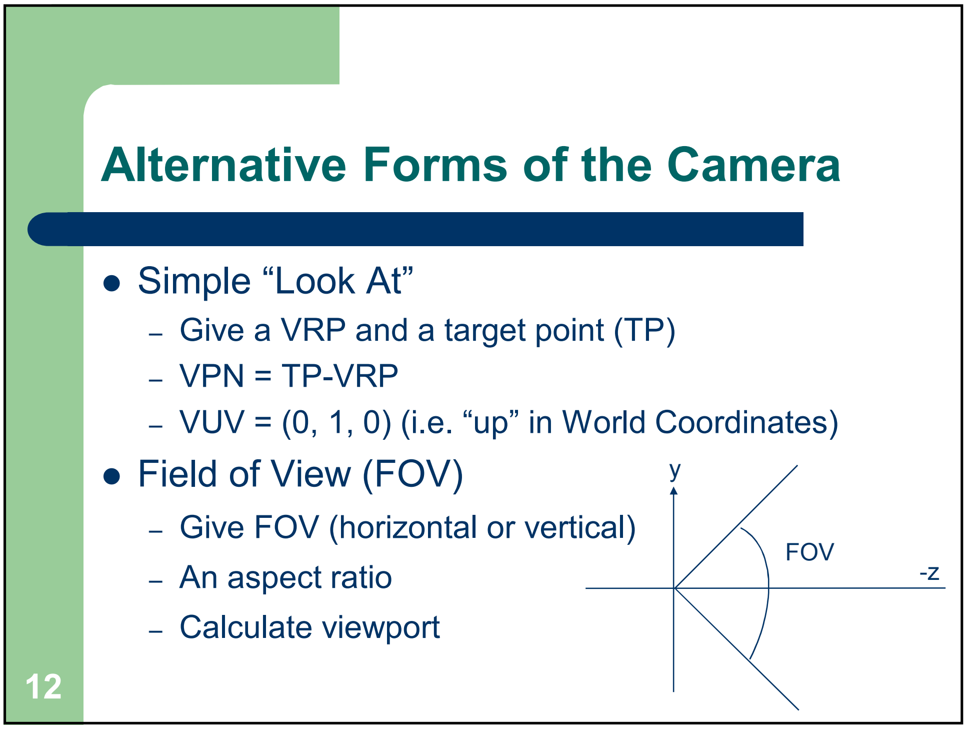
COP VPN VPWindow VPDistance

Resolution

A view consists of a camera with settings controlled by parameters

11

Dr. Y. Hu



Alternative Forms of the Camera

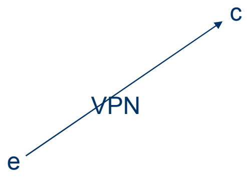
- Simple “Look At”
  - Give a VRP and a target point (TP)
  - VPN = TP-VRP
  - VUV = (0, 1, 0) (i.e. “up” in World Coordinates)
- Field of View (FOV)
  - Give FOV (horizontal or vertical)
  - An aspect ratio
  - Calculate viewport

12

OpenGL - glu Viewing

- Constructing an ‘M’ matrix

```
gluLookAt(ex,ey,ez,    //eye = COP (world coord.)
          cx,cy,cz,    //point of interest
          upx,upy,upz //up vector
          )
```



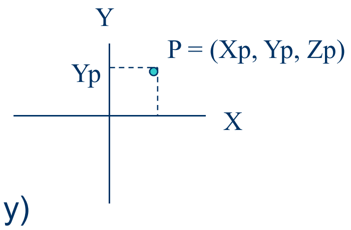
- Matrix that maps
  - `(cx,cy,cz)` to –Z axis
  - `(ex,ey,ez)` becomes the origin
  - `(upx,upy,upz)` becomes the y-axis
- Pre-multiplies current matrix

Projections

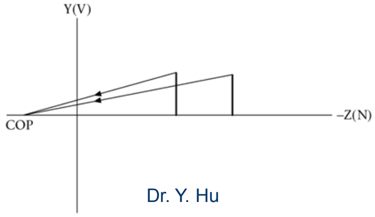
- Orthographic projection

  - COP at infinite far from the view plane

  - Z = 0 and point P = (X, Y, Z) projects to image point p = (x, y) where x = X and y = Y

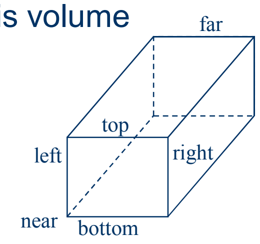


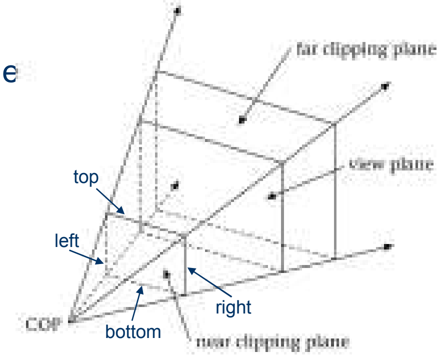
- Perspective projection



View Volume

- View volume = Clipping volume
- **Clipping volume**
  - The region of the scene
  - Not rendering outside this volume

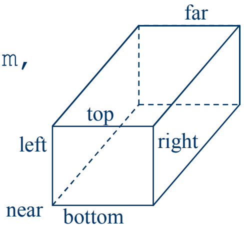




The Clipping Volume - Ortho

- In 3D

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



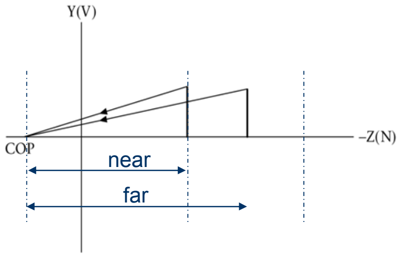
- In 2D

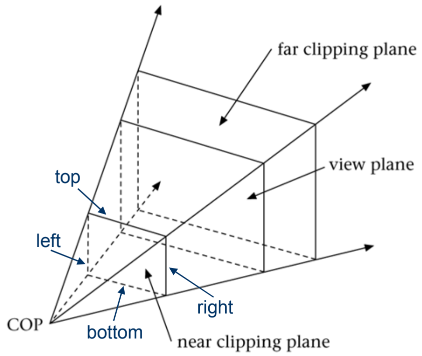
```
void glOrtho2D(left, right, bottom, top);
```

(glOrtho2D sets near and far to -1.0 and 1.0 respectively)

The Clipping Volume - Perspective

- **Frustrum**





```
glFrustrum(GLdouble left, GLdouble right, GLdouble
bottom, GLdouble top, GLdouble near, GLdouble far)
```

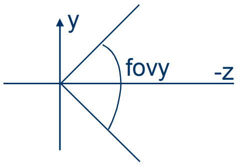
Dr. Y. Hu

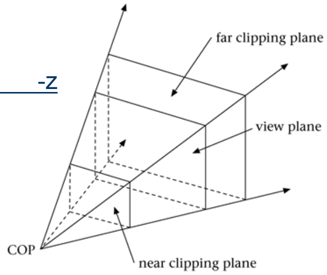
17

OpenGL - glu Perspective

- glu projection matrix:

```
gluPerspective(fovy,    //field of view degrees
               aspect, //xwidth/yheight
               zNear,  //near clipping plane
               zFar   //far clipping plane
              )
```





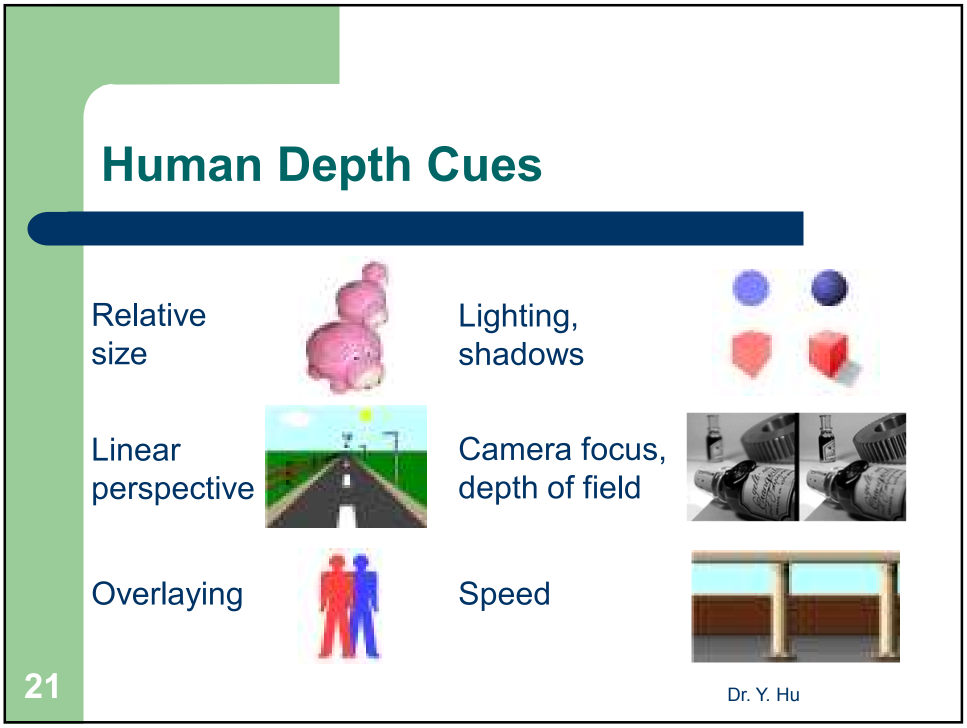
- Same viewing volume as `glFrustrum()`

18

Revisit: Viewport

- Clipping volume ← **world coordinates**.
- OpenGL rendering ← **screen coordinates**.
- The drawing region within the clipping volume on a screen → **`ViewPort`.**

Stereo Viewing



Human Depth Cues

Relative size

Lighting, shadows

Linear perspective

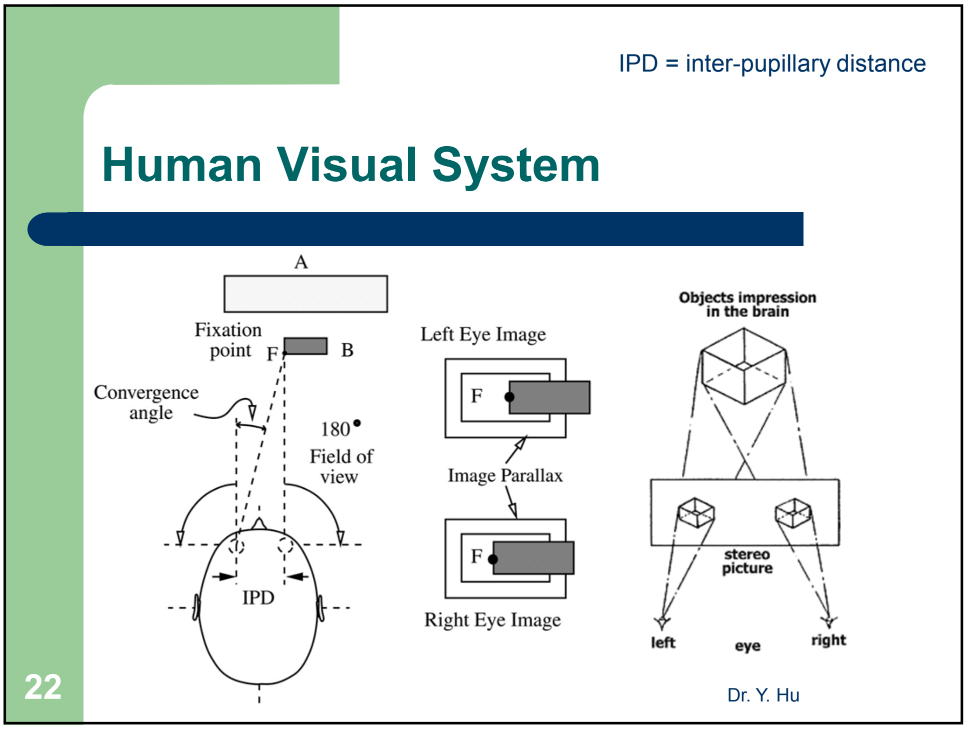
Camera focus, depth of field

Overlaying

Speed

21

Dr. Y. Hu



IPD = inter-pupillary distance

Human Visual System

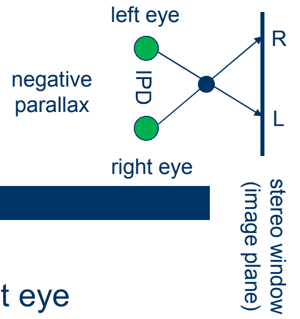
22

Dr. Y. Hu

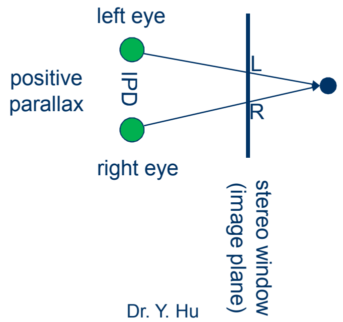
Terminology

- Accommodation
  - → Adjustment of the focal length of the eyes
- Convergence
  - → Eye rotation inwards and parallel
- Binocular disparity
  - → Image differences produced by left and right eyes
- Motion parallax
  - → Relative movement of points with respect to head

Stereoscopy



- Stereo pairs
  - → Two projections, left and right eye on flat display
- Horizontal parallax (R – L)
  - R-L < 0; negative (pop-up)
  - R-L > 0; positive
- Similar term for vertical parallax



Viewing Stereo Pairs

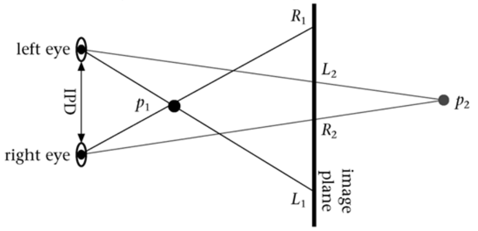
- Uncrossed/parallel setup

  → when right eye sees right image and left eye the left image

- Crossed setup

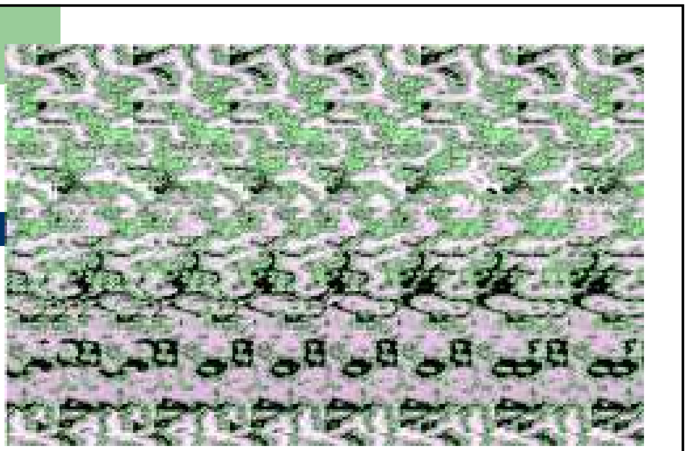
  → when right eye sees left image and left eye sees right image

- How to reverse the sense of depth?



25

Try!!

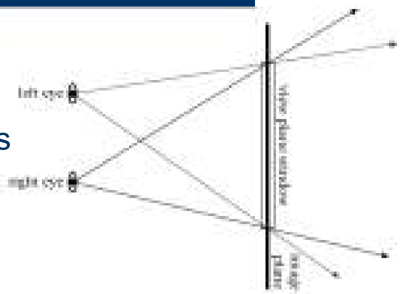


http://www.3dartist.com/3dao/stereo.htm

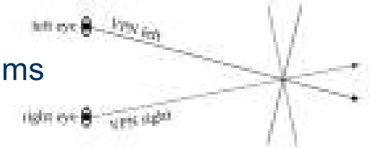
26

Stereo Viewing - Programming

- Two cameras
  - → Each eye = one camera
  - → Set left and right cameras

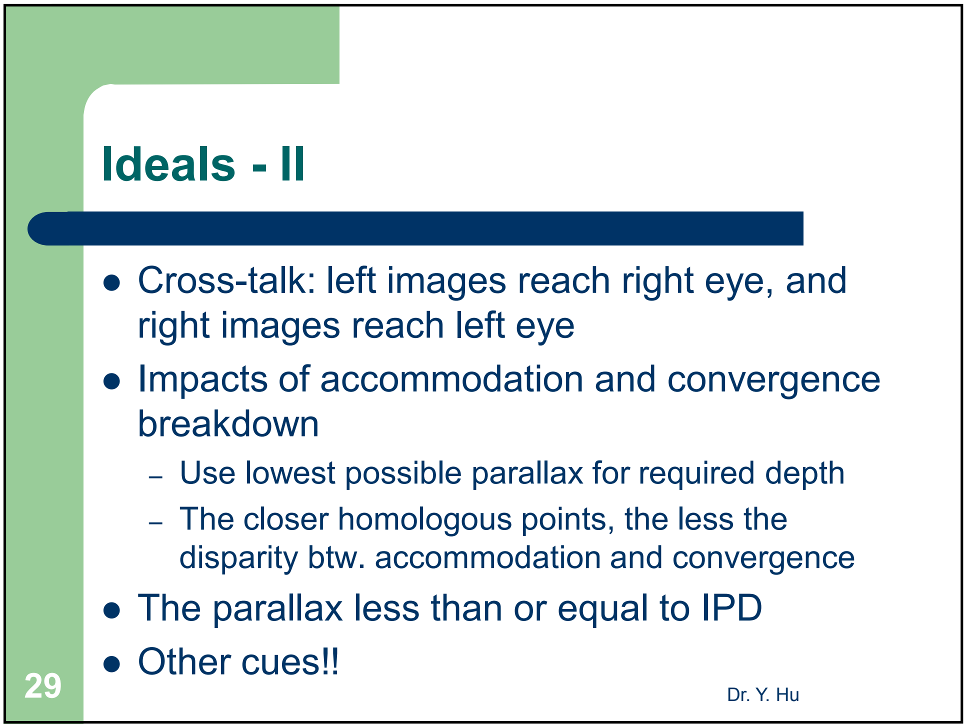


- One camera
  - → Each eye = one frustum
  - → Draw left and right frustums



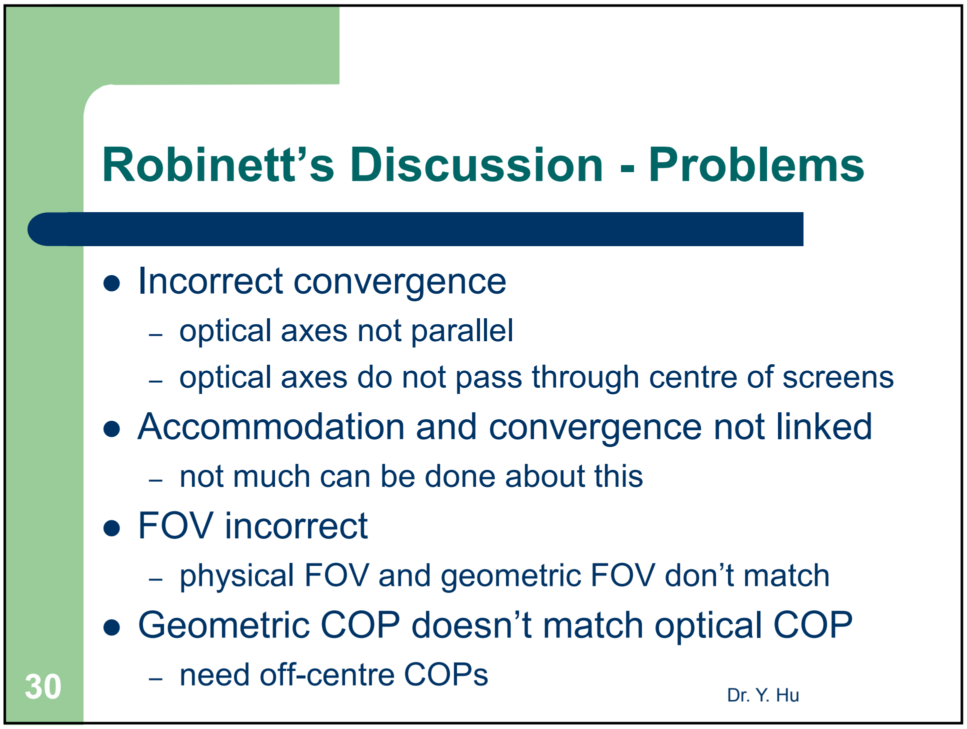
Ideals - I

- Congruence for left and right images
  - → colour, geometry, brightness
- Avoidance of vertical parallax (should be zero)
- Wide parallax (large separation of the eyes)
  - → good depth, but discomfort
- Maximum depth, but lowest parallax
- Further distance of viewer from display, the greater the parallax that can be tolerated



Ideals - II

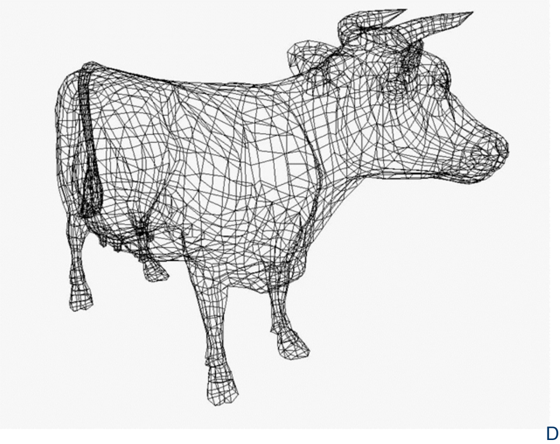
- Cross-talk: left images reach right eye, and right images reach left eye
- Impacts of accommodation and convergence breakdown
  - Use lowest possible parallax for required depth
  - The closer homologous points, the less the disparity btw. accommodation and convergence
- The parallax less than or equal to IPD
- Other cues!!



Robinett’s Discussion - Problems

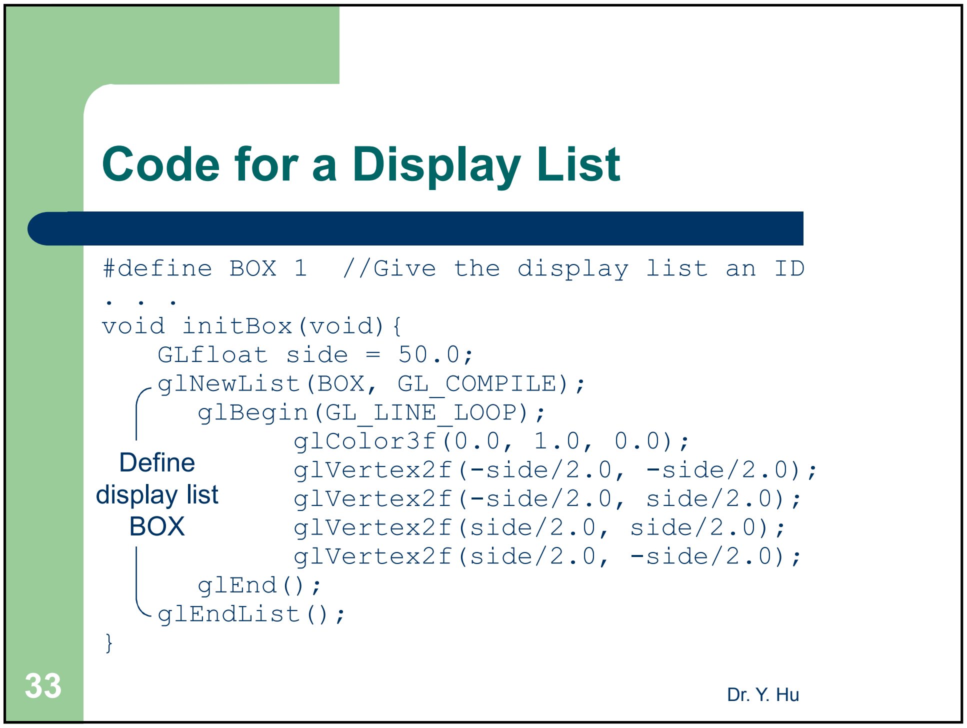
- Incorrect convergence
  - optical axes not parallel
  - optical axes do not pass through centre of screens
- Accommodation and convergence not linked
  - not much can be done about this
- FOV incorrect
  - physical FOV and geometric FOV don’t match
- Geometric COP doesn’t match optical COP
  - need off-centre COPs

Complex 3D Object



Display Lists

- Until now, draw objects as define them.
  - ➔ Once drawn, no way to refer to them again.
- To define a single object and refer to it later.
  - ➔ In OpenGL, use a **display list.**



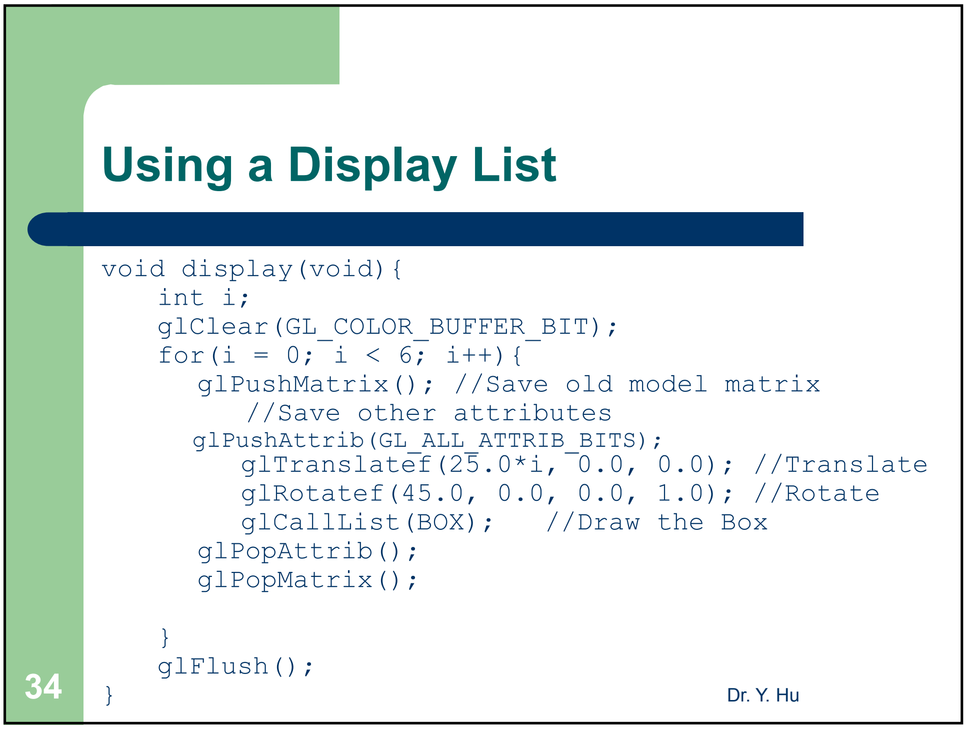
Code for a Display List

```
#define BOX 1  //Give the display list an ID
. . .
void initBox(void){
    GLfloat side = 50.0;
    glNewList(BOX, GL_COMPILE);
      glBegin(GL_LINE_LOOP);
            glColor3f(0.0, 1.0, 0.0);
            glVertex2f(-side/2.0, -side/2.0);
            glVertex2f(-side/2.0, side/2.0);
            glVertex2f(side/2.0, side/2.0);
            glVertex2f(side/2.0, -side/2.0);
      glEnd();
    glEndList();
}
```

Define display list BOX

33

Dr. Y. Hu



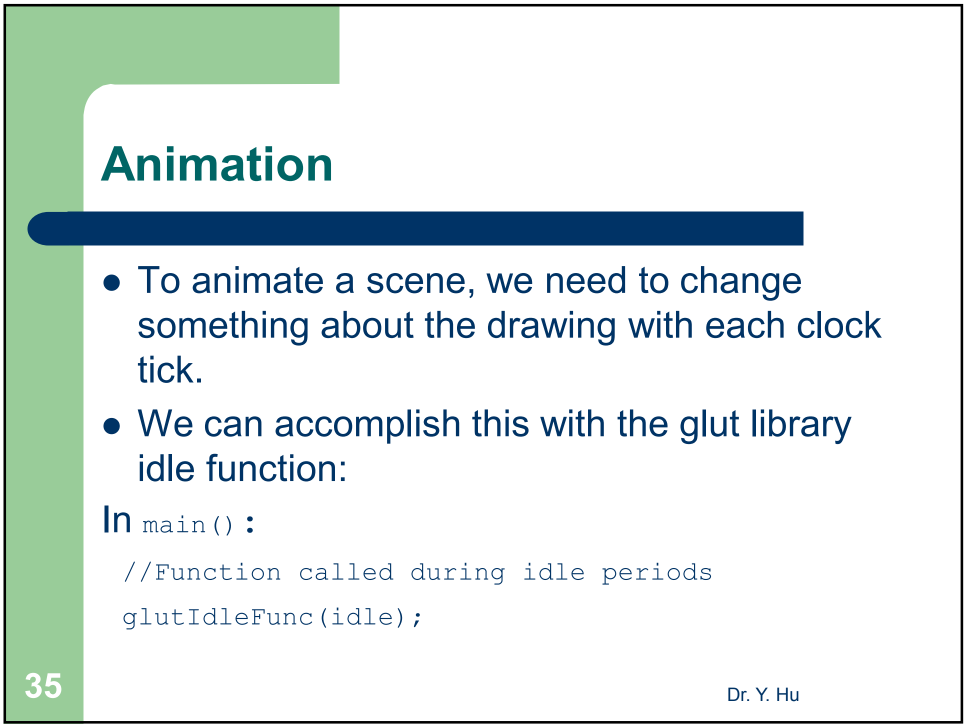
Using a Display List

```
void display(void){
    int i;
    glClear(GL_COLOR_BUFFER_BIT);
    for(i = 0; i < 6; i++){
      glPushMatrix(); //Save old model matrix
         //Save other attributes
      glPushAttrib(GL_ALL_ATTRIB_BITS);
         glTranslatef(25.0*i, 0.0, 0.0); //Translate
         glRotatef(45.0, 0.0, 0.0, 1.0); //Rotate
         glCallList(BOX);   //Draw the Box
      glPopAttrib();
      glPopMatrix();

    }
    glFlush();
}
```

34

Dr. Y. Hu

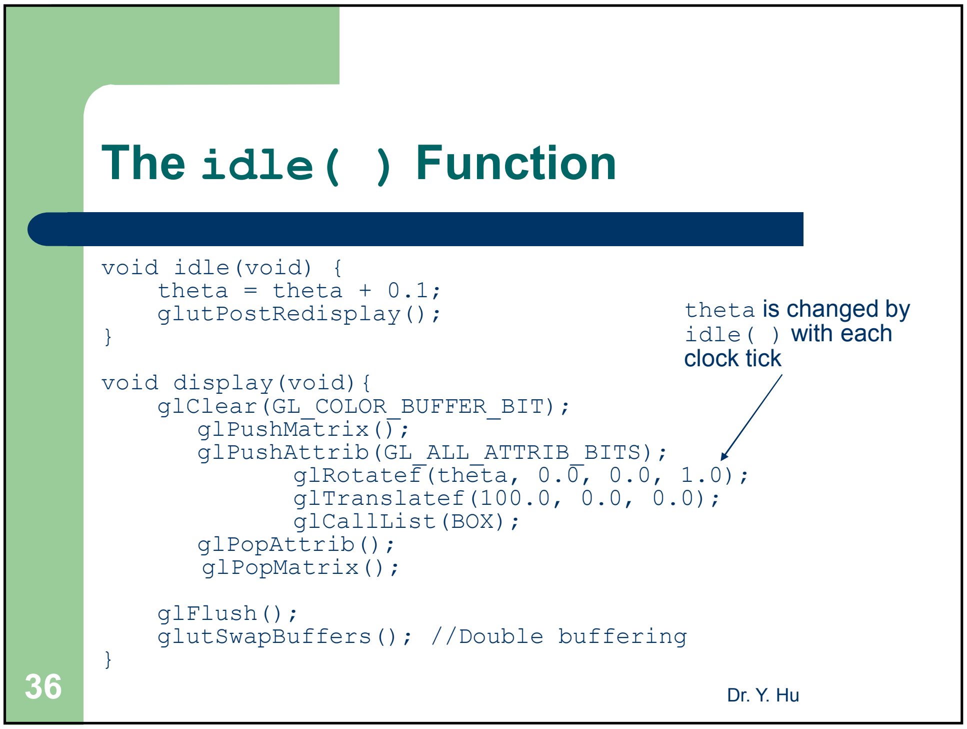


Animation

- To animate a scene, we need to change something about the drawing with each clock tick.
- We can accomplish this with the glut library idle function:

In `main()`:

```
//Function called during idle periods
glutIdleFunc(idle);
```



The `idle( )` Function

```
void idle(void) {
    theta = theta + 0.1;
    glutPostRedisplay();
}

void display(void){
    glClear(GL_COLOR_BUFFER_BIT);
       glPushMatrix();
       glPushAttrib(GL_ALL_ATTRIB_BITS);
              glRotatef(theta, 0.0, 0.0, 1.0);
              glTranslatef(100.0, 0.0, 0.0);
              glCallList(BOX);
       glPopAttrib();
       glPopMatrix();

    glFlush();
    glutSwapBuffers(); //Double buffering
}
```

`theta` is changed by `idle( )` with each clock tick

Double Buffering

- Draw into one buffer while displaying the other, then swap the two.
- To guarantee that a scene is displayed only after the drawing is finished.

In `main()`:

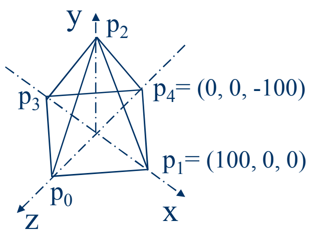
```
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
```

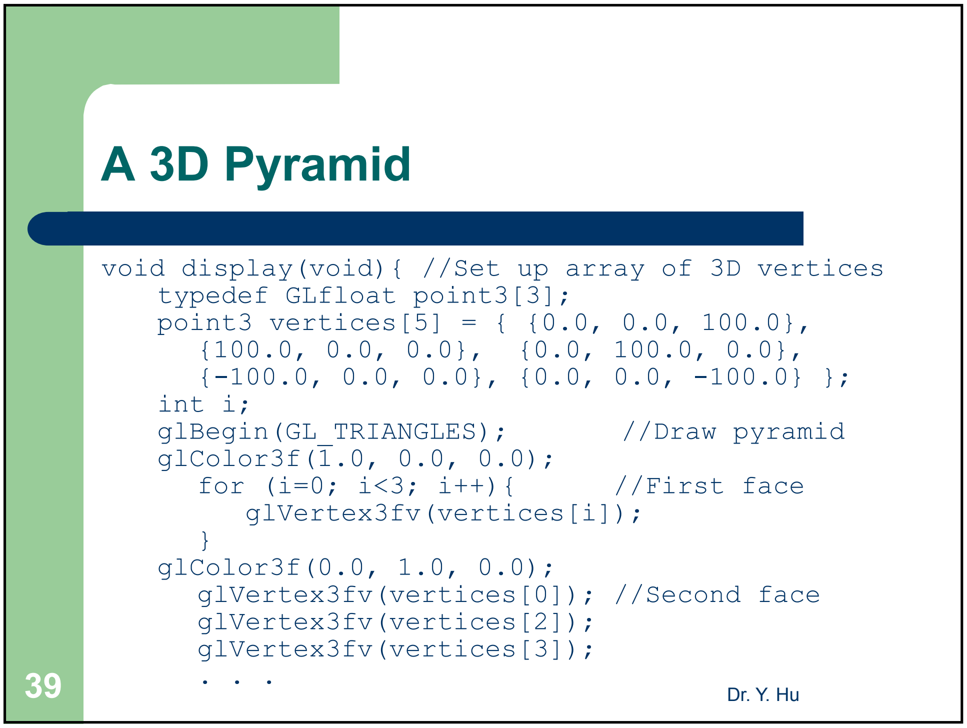
In `display()`:

```
glutSwapBuffers(); //Double buffering
```

Drawing in 3D

- Use 3D points instead of 2D points.
- Example: 3D pyramid.



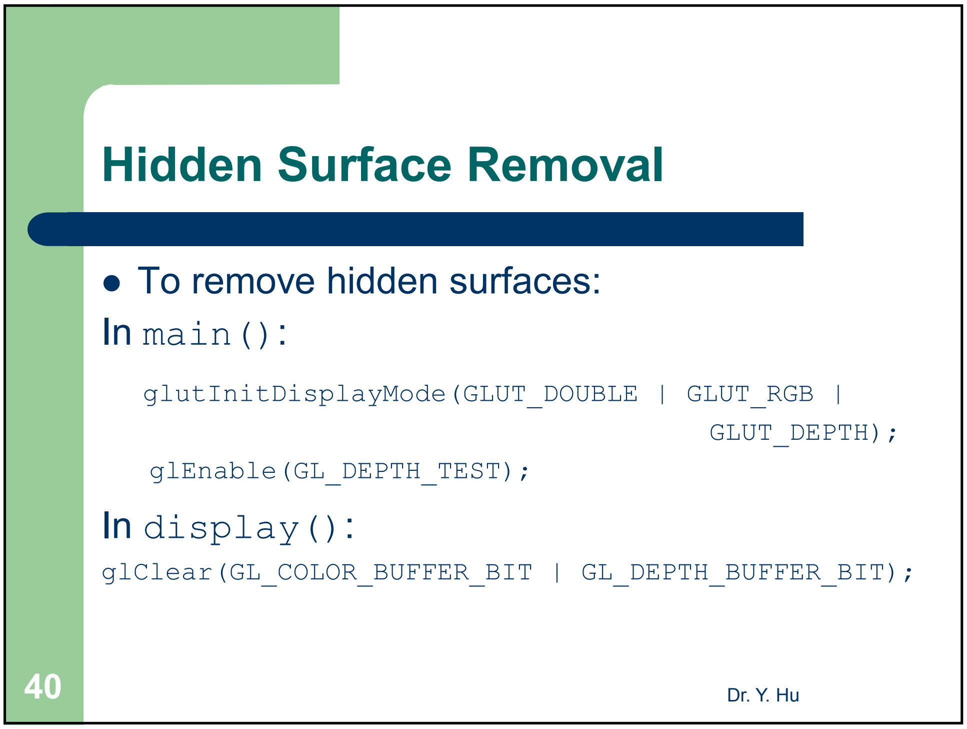


A 3D Pyramid

```
void display(void){ //Set up array of 3D vertices
   typedef GLfloat point3[3];
   point3 vertices[5] = { {0.0, 0.0, 100.0},
      {100.0, 0.0, 0.0},  {0.0, 100.0, 0.0},
      {-100.0, 0.0, 0.0}, {0.0, 0.0, -100.0} };
   int i;
   glBegin(GL_TRIANGLES);       //Draw pyramid
   glColor3f(1.0, 0.0, 0.0);
      for (i=0; i<3; i++){      //First face
         glVertex3fv(vertices[i]);
      }
   glColor3f(0.0, 1.0, 0.0);
      glVertex3fv(vertices[0]); //Second face
      glVertex3fv(vertices[2]);
      glVertex3fv(vertices[3]);
      . . .
```

39

Dr. Y. Hu



Hidden Surface Removal

- To remove hidden surfaces:

In `main()`:

```
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB |
                                    GLUT_DEPTH);
   glEnable(GL_DEPTH_TEST);
```

In `display()`:

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

40

Dr. Y. Hu

Vertex Arrays

- A vertex array specifies the order of vertices to be drawn in a given set of drawing commands.
- To access it, use a pointer to the index of the first vertex to be drawn.

In `main()`:

```
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3, GL_FLOAT, 0, vertices);
```

Setting Up a Vertex Array

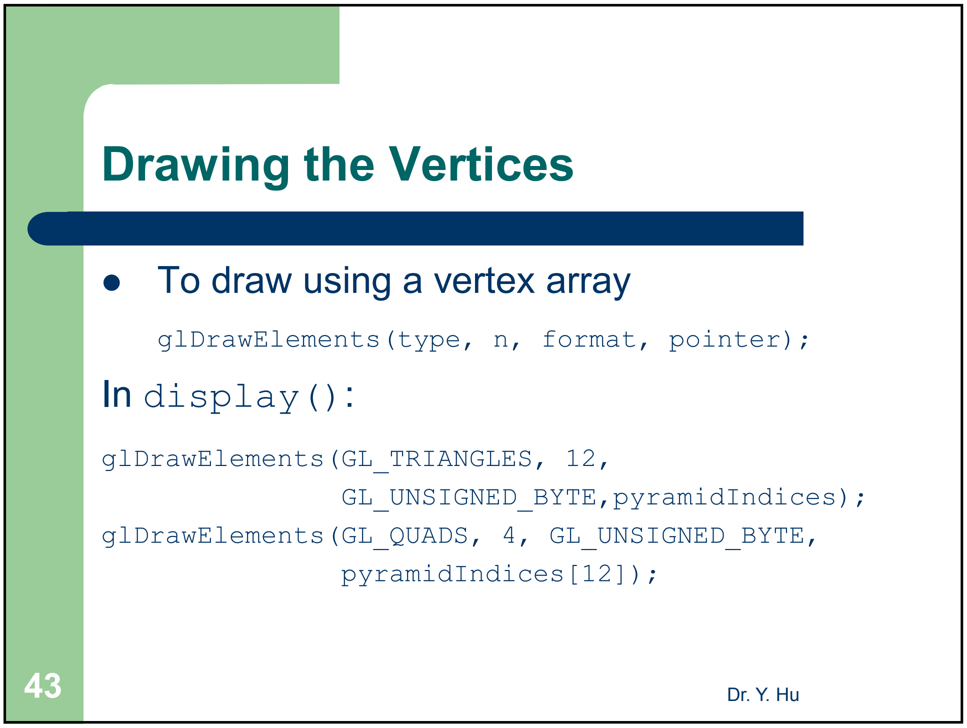
- Create an array of all vertices needed:

$$\texttt{GLfloat vertices[]=\{}\underbrace{\texttt{0.0,0.0,100.0}}_{p_0}\texttt{, }\underbrace{\texttt{100.0,0.0,0.0}}_{p_1}\texttt{,}$$

$$\underbrace{\texttt{0.0,100.0,0.0}}_{p_2}\texttt{, }\underbrace{\texttt{-100.0,0.0,0.0}}_{p_3}\texttt{, }\underbrace{\texttt{0.0,0.0,-100.0}}_{p_4}\texttt{\};}$$

- Provide an ordered list of vertices by index in the array that specifies each polygon to be drawn. A 3D pyramid, requires 16 vertices altogether.

```
 GLubyte pyramidIndices[16] = {0, 1, 2, 0, 2, 3, 3,
   2, 4, 4, 2, 1, 0, 3, 4, 1}
```



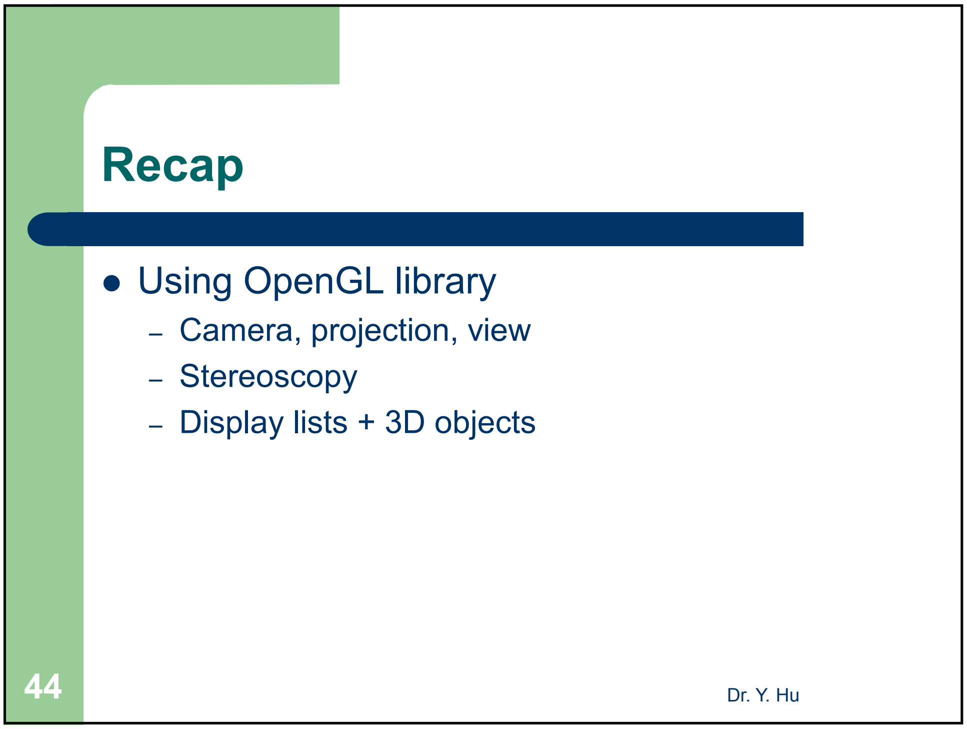
Drawing the Vertices

- To draw using a vertex array

```
glDrawElements(type, n, format, pointer);
```

In `display()`:

```
glDrawElements(GL_TRIANGLES, 12,
               GL_UNSIGNED_BYTE,pyramidIndices);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE,
               pyramidIndices[12]);
```



Recap

- Using OpenGL library
  - Camera, projection, view
  - Stereoscopy
  - Display lists + 3D objects