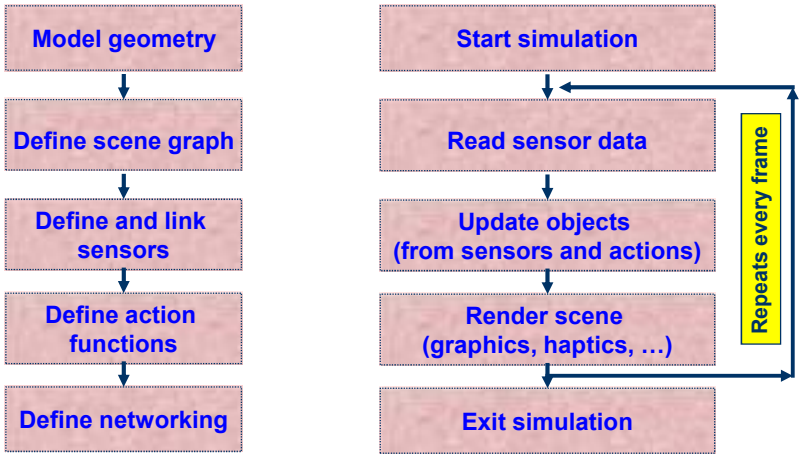
ENSF 545
Introduction to Virtual Reality

VR Programming

How to Build a VR Simulation?

2

Authoring Stage + Run-time Loop



3

Major Concepts of Graphics

- Separation of
  - Scene → Scene is independent of any view
  - Viewing → Views are unconstrained
  - Rendering → There are many possible rendering methods given a scene and a view

4

Dr. Y. Hu

(Partial) Scene

What Does Graphics API's Do?

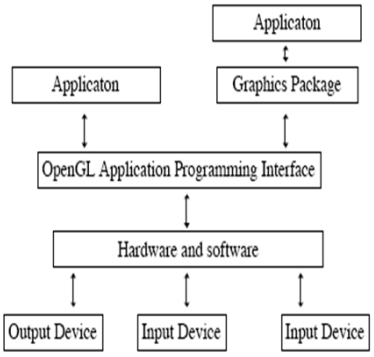
- Objects ← **primitives**
- Primitives ← **attributes** (color, materials etc.)
- **Transformations** ← translations, rotations
- **Viewing** ← camera position, orientation, etc.
- **Input** mechanisms ← user interface
- **Control** ← communicating with operating systems, initializing programs, etc.

VR Graphics and Utility Libraries

- Low level
  - **OpenGL**, Direct 3D
- User interface libraries
  - **Glut**, Tweek
- Specific purposes
  - 3DGame Studio
  - PeopleShop
  - **OpenHaptics**
  - DirectSound 3D
- High level
  - Open Inventor
  - Coin 3D
  - Open Performer
  - Open SG
  - VR Juggler
  - Java 3D
  - VRML
  - WorldToolKit (WTK)
  - VTK

OpenGL – www.opengl.org

- Developed in early 90s
- Standardized
  - Managed by Architectural Review Board (ARB)
- Procedural model
- Independent of operating systems



OpenGL Libraries

- **Gl & Glu:** The OpenGL utilities.
  - Containing code for OpenGL functions, definitions.
- **Glut:** The OpenGL utility toolkit.
  - For using OpenGL within a windowing environment (e.g. windows or Mac OS).
  - Operating system dependent.

OpenGL Model and Process

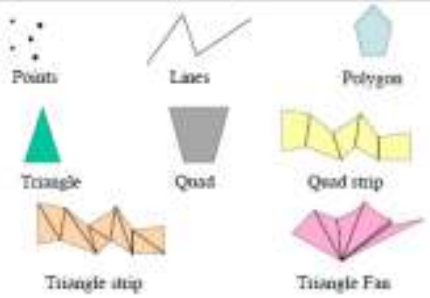
- Model:
  - Define 3D objects in space.
  - Specify camera properties (position, orientation, projection system, etc).
- Process:
  - Transformation
  - Clipping
  - Projection
  - Rasterization

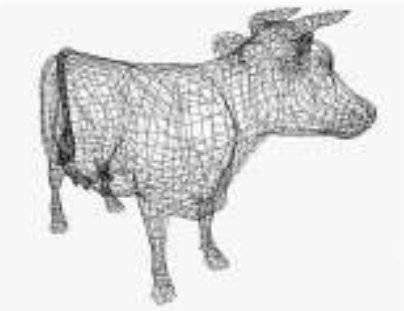
OpenGL Basics

- Draw primitives (lines, polygons)
- Draw 3D objects
- Use a synthetic camera to form images
- Convention:

```
glFunction(para…);
GLtype
GL_TRIANGLES
```

Primitives, Polygonal Meshes



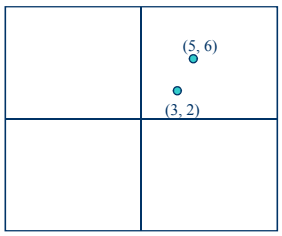


13

OpenGL primitives: Vertices

- OpenGL defines objects in terms of vertices (points).
  - 2D shapes: polygons defined by vertices.
  - 3D shapes: groups of polygons.
- Example:

```
glBegin(GL_POINTS);
   glVertex2f(3.0, 2.0);
   glVertex2f(5.0, 6.0);
glEnd();
```



2D plane

Dr. Y. Hu

14

Defining a Vertex

- A 2D vertex:

```
glVertex2f(GLfloat x, GLfloat y);
```

2D vertex floating point openGL parameter type

- A 3D vertex:

```
glVertex3f(GLfloat x, GLfloat y, GLfloat z);
```

- **Example**:
  - A 3D vertex using integer values ??

Arrays → Define Points

```
GLfloat myVertex[3];
myVertex[0] = 4.0;            //x value
myVertex[1] = 2.0;            //y value
myVertex[2] = 1.0;            //z value

  . . .
glVertex3fv(myVertex);
```

using array

pointer to array
containing point values

typedef → Define Points

Array of points as 2D array:

```
GLfloat myPoints[5][2]; //array of 5 2D points
```

Using typedef:

```
//a new type, point2, as an array of 2 GLfloats
typedef GLfloat point2[2];

point2 myPoint;        //a 2 element array
myPoint[0] = 1.0;
myPoint[1] = 2.7;
```


An Array of Points

```
//a new type, point2, as an array of 2 GLfloats
typedef GLfloat point2[2];
point2 myPoints[4] =
  {{1.0, 3.5}, {4.2, 6.7}, {3.1, 2.2}, {7.2, 1.1}};
```

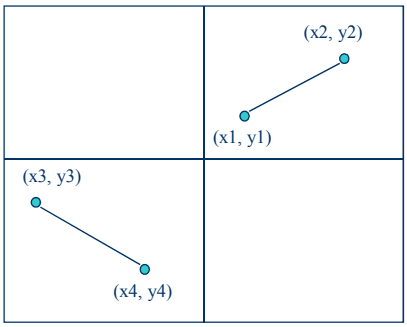
What is the value of:

- `myPoints[0]`
- `myPoints[2][1]`

Connect Points

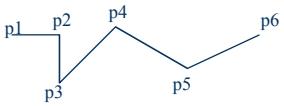
GL_LINES connects **pairs of points**.

```
glBegin(GL_LINES);
  glVertex2f(x1, y1);
  glVertex2f(x2, y2);
  glVertex2f(x3, y3);
  glVertex2f(x4, y4);
glEnd();
```

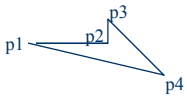


Other Drawing Functions

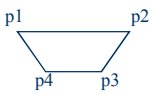
GL_LINE_STRIP:



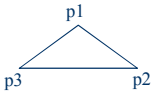
GL_LINE_LOOP:



GL_POLYGON:

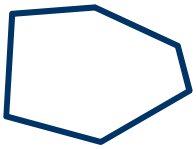


GL_TRIANGLES:

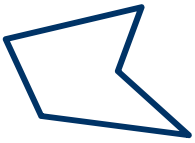


Convex, Concave

- Convex



- Concave

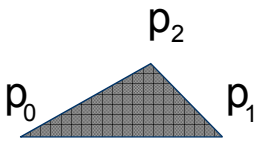


Normal: Order of Vertices

- The cross product

$$n = (p_1 - p_0) \times (p_2 - p_0)$$

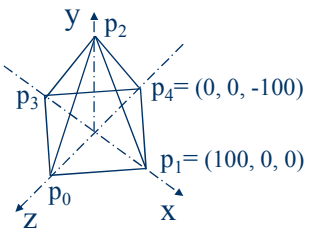
defines a **normal** to the plane



- **Normal**: A vector perpendicular to a plane
  - Pointing outward of the positive face of the plane
  - Important for lighting and shading of objects

Object in 3D

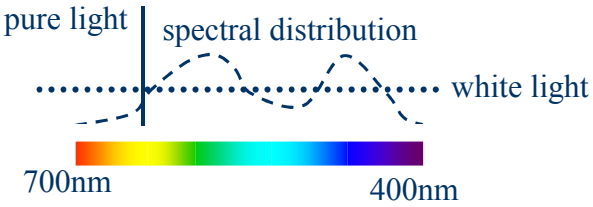
- Use 3D points instead of 2D points.
- Example: 3D pyramid.



Dr. Y. Hu

Light in an Environment

- Light:



- Reflection:

Dr. Y. Hu

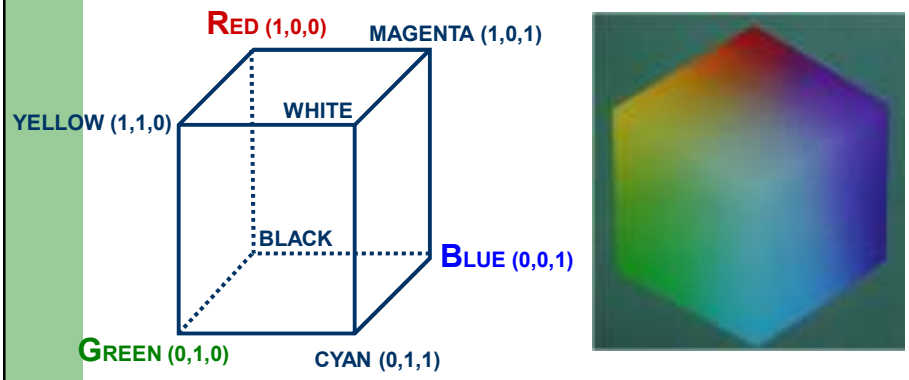
The Frame Buffer

- The **frame buffer** stores the value of each pixel in the viewing window.
- Each pixel has a given number of bits to encode the color. The number of bits is the **bit depth**.
  - 8-Bit depth → 256 possible colors (2^8)
  - 32-Bit depth →millions of possible colors (2^{32})
  - 64-Bit depth → ???

Indexed Color

OpenGL:
Don't depend on particular hardware or number of bits per pixel.
Use generic color scale between 0 and 1.0 for each R, G, B value.

RGB Color Model



OpenGL Shading and Color

- Shading properties

```
glShadeModel(GL_SMOOTH | GL_FLAT)
```

- Color

```
glColorNT{V}(r,g,b,{α})
```

  - N = 3, 4
  - T = f, b, s, i, ub, ui, us
  - v implies passing a pointer to array of color

OpenGL Color Specification

```
glDisable (GL_LIGHTING);
glColor3f(r, g, b); //r, g, b value btw.0.0 and 1.0
   … …
glEnable (GL_LIGHTING);

glColor3f(1.0, 0.0, 0.0);      //What color is this?
glColor3f(1.0, 0.0, 1.0);
glColor3f(0.0, 1.0, 0.0);
```

α Channel: - a 4th color parameter
- opacity (1.0), transparency (0.0)

```
glClearColor(1.0, 1.0, 1.0, 1.0);
```

RGB white

α opaque

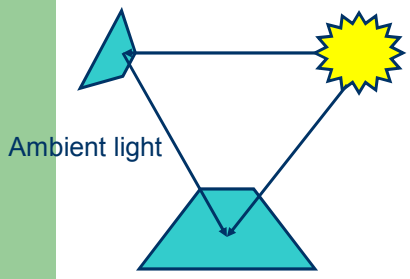
OpenGL Materials

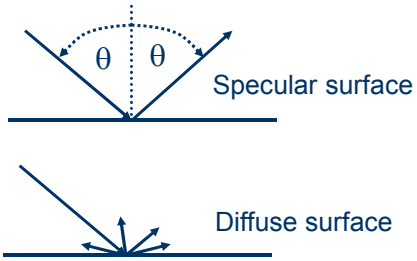
- Many lighting parameters
- Specify a material as
  - ambient, shininess, diffuse, specular

```
GLfloat mat_spec = { 0.5, 0.5, 1.0, 1.0};
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_spec)
glColorMaterial(GL_FRONT, GL_DIFFUSE)
```

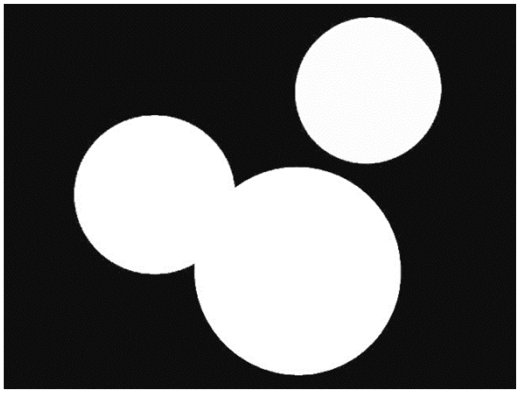
Lighting Model

- Any point in the environment receives light from all around





The Image - Detection



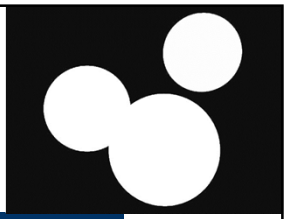
Ambient Light

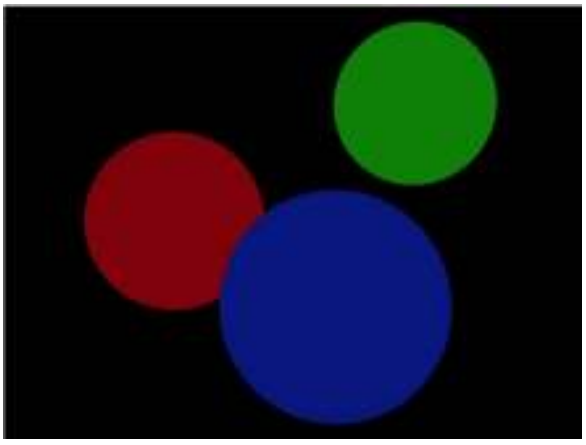
- Approximation to global illumination
  - Illuminates each object to a certain extent
  - Be constant across a whole object
- Usually set for whole scene (I_a)
- Each object reflects only a proportion (k_a)
- So far then

$$I_r = k_a I_a$$

- But using RGB, so .... ?

The Image - Ambient

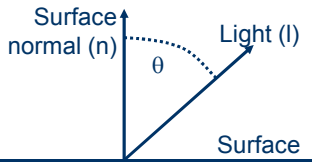




Diffuse Light - Lambert's Law

- Lambert's law: Reflected intensity is proportional to *cosθ*
- The proportion of light reflected due to Lambert's law rather than absorbed ($k_d \rightarrow$ as $k_{d,red}$ $k_{d,green}$ and $k_{d,\ blue}$)
- Light with ambient and diffuse components

$$I_r = k_a I_a + k_d I_i (n.l)$$



Multiple Lights?

- Add the diffuse terms

$$I_r = k_a I_a + \sum_{j=1}^{m} k_d I_{i,j} (n.l_j)$$

- $I_{i,j}$ is the incoming intensity of light j
- l_j is the vector to light j

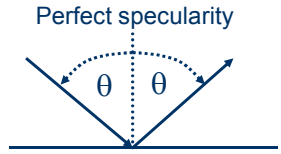
The Image - Diffuse

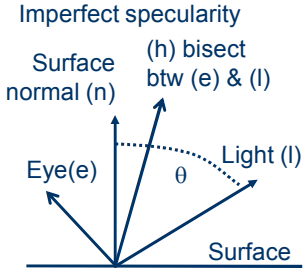
Imperfect Specularity (Phong)

- Specular component:

$$k_s I_i (h.n)^m$$

- m is the power of the light
  - High m implies smaller specular highlight
  - Low m makes the highlight more blurred





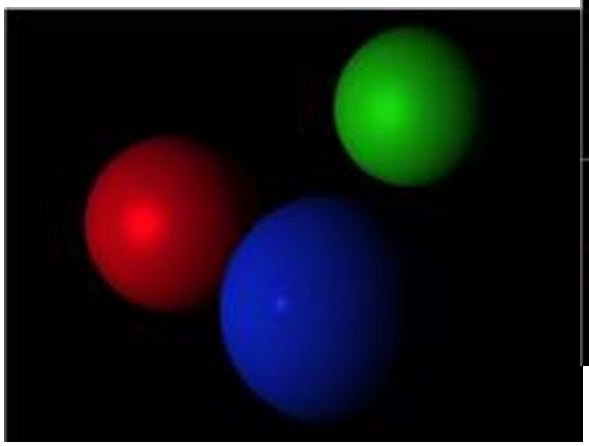
Specular Light

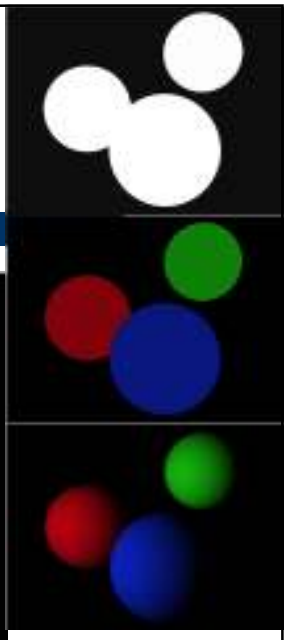
- Ambient, diffuse and specular components

$$I_r = k_a I_a + I_i (k_d (n.l) + k_s (h.n)^m)$$

- If multiple lights ➔ ????

The Image - Specular





OpenGL Light Models

- Light Models:

```
GL_LIGHT_MODEL_AMBIENT          (Ambient RGBα intensity)
GL_LIGHT_MODEL_Local_VIEWER     (Specular reflection)
GL_LIGHT_MODEL_TWO_SIDE         (Lighting 1-sided or 2-sided)
```

- Example:

```
GLfloat lmodel_ambient [] = {0.2, 0.2, 0.2, 1.0}
glLightModelfv(GL_LIGHT_MODEL_AMBIENT,lmodel_ambient)
GLfloat light_pos ={ 1.0, 2.0, 1.0, 0.0}
glLightfv(GL_LIGHT0, GL_POSITION, light_pos)
glEnable(GL_LIGHTING)
glEnable(GL_LIGHT0)
```


Concatenation of Transformations

- Suppose you want to
  - Scale an object: P1 = SP
  - Rotate the object: P2 = RP1 = RSP
  - Translate the object: P3 = TP2 = TRSP
- With a new matrix M = TRS, P3 = MP

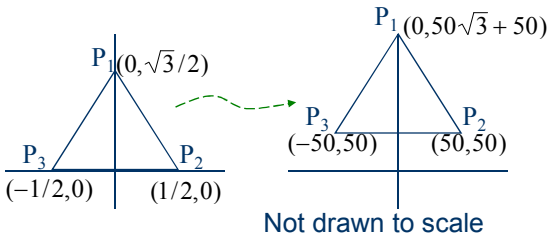
Transformations in OpenGL

- Creating your own transformation:
  - Computing transformation in advance
  - Entering matrix into array
  - Loading the matrix
  - Drawing object
- Use the OpenGL transformation functions

Computing Transformation

Example:

Scale by 100 in both the x and y directions, then translate vertically by 50.



Not drawn to scale

Entering Matrix into Array

In OpenGL, a transforming matrix is stored as a 16 element array in **column major order.**

$$\begin{pmatrix} 0 & 4 & 8 & 12 \\ 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \end{pmatrix}$$

```
scaleMatrix[0] = 100.0;
scaleMatrix[5] = 100.0;
scaleMatrix[10] = 1.0;
scaleMatrix[13] = 50.0;
scaleMatrix[15] = 1.0;
```

$$M = \begin{pmatrix} 100 & 0 & 0 & 0 \\ 0 & 100 & 0 & 50 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

47

Dr. Y. Hu

Loading Matrix + Drawing Object

1. Make sure the correct matrix mode
   ```
   glMatrixMode(GL_MODELVIEW);
   ```
2. Push the current model matrix onto the matrix stack
   ```
   glPushMatrix();
   ```
3. Load in the transformation matrix
   ```
   glLoadMatrixf(scaleMatrix);
   ```
4. Draw the object for transform
   ```
   glBegin(GL_LINE_LOOP);
         glVertex2f(x, y);
         ...
   glEnd();
   ```
5. Pop the old model matrix off the stack
   ```
   glPopMatrix();
   ```

48

Dr. Y. Hu

Example Code

```
for(i = 0; i<16; i++){      //Set matrix to zero
    scaleMatrix[i] = 0;
}
scaleMatrix[0] = 100.0;     //Scale x by 100 fold
scaleMatrix[5] = 100.0;     //Scale y by 100 fold
scaleMatrix[10] = 1.0;      //Keep Z constant
scaleMatrix[13] = 50.0;     //Translate in y by 50.0
scaleMatrix[15] = 1.0;      //This element is always 1

glPushMatrix();             //Store the current matrix
glLoadMatrixf(scaleMatrix); //Load the scale matrix
glBegin(GL_LINE_LOOP);         //Draw the triangle
    for (i=0; i<3; i++){
       glVertex2fv(triVerts[i]);
}
glEnd();
glPopMatrix();      //Get original matrix back
```


OpenGL Transform Functions

- Translate by vector (dx, dy, dz)

```
glTranslatef(dx, dy, dz);
```

- Rotate by angle about axis by (vx, vy, vz)

```
glRotatef(angle, vx, vy, vz);
```

- Scale by factor given by (sx, sy, sz)

```
glScalef(factor, sx, sy, sz);
```

Order of Transformations

- Suppose: transformations in the order of:
  1. Scale 2. Rotate 3. Translate
- Recall concatenation of matrix multiplications:
  P' = M*P; M = ??
- Start with the identity matrix (I):
  1. Call translatef() M = I*T
  2. Call rotatef() M = I*T*R
  3. Call scalef() M = I*T*R*S
  4. Draw P P' = MP = T*R*S*P

Example Code

```
   glPushMatrix();            //Save model matrix
//Scale by 100 in x and y, done last
   glScalef(100.0, 100.0, 1.0);
//Translate in x direction, done 3rd
   glTranslatef(-sqrt(3.0), 0.0, 0.0);
//Rotate by 90 deg around z axis, done 2nd
   glRotatef(-90.0, 0.0, 0.0, 1.0);
//Scale along y axis only, done 1st
   glScalef(1.0, 2.0, 1.0);
   glBegin(GL_LINE_LOOP); //Draw a triangle
      for (i=0; i<3; i++){
         glVertex2fv(triVerts[i]);
      }
   glEnd();
   glPopMatrix(); //get original model matrix back
```

Concatenating Transformation Matrix

- Concatenate a transformation matrix with other transformations, use the openGL function:

```
glMultMatrixf(const GLfloat *m);
```

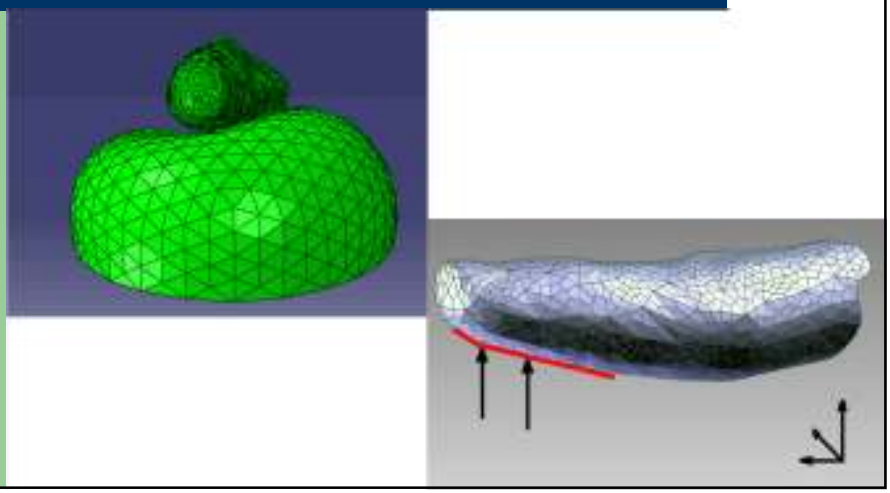
- Example: Concatenating the `scaleMatrix` with a rotation of 90 degrees (assume `scaleMatrix` has been initialized to desired values):

```
glRotatef(-90.0, 0.0, 0.0, 1.0);
glMultMatrixf(scaleMatrix);
glBegin(GL_LINE_LOOP);
       ...
```


Recap

- Using OpenGL library
  - Geometric primitives + 3D objects
  - Color + Lighting
  - Transformation

Polygon Partitioning Object



55