Introduction to OpenGL Programming (2)

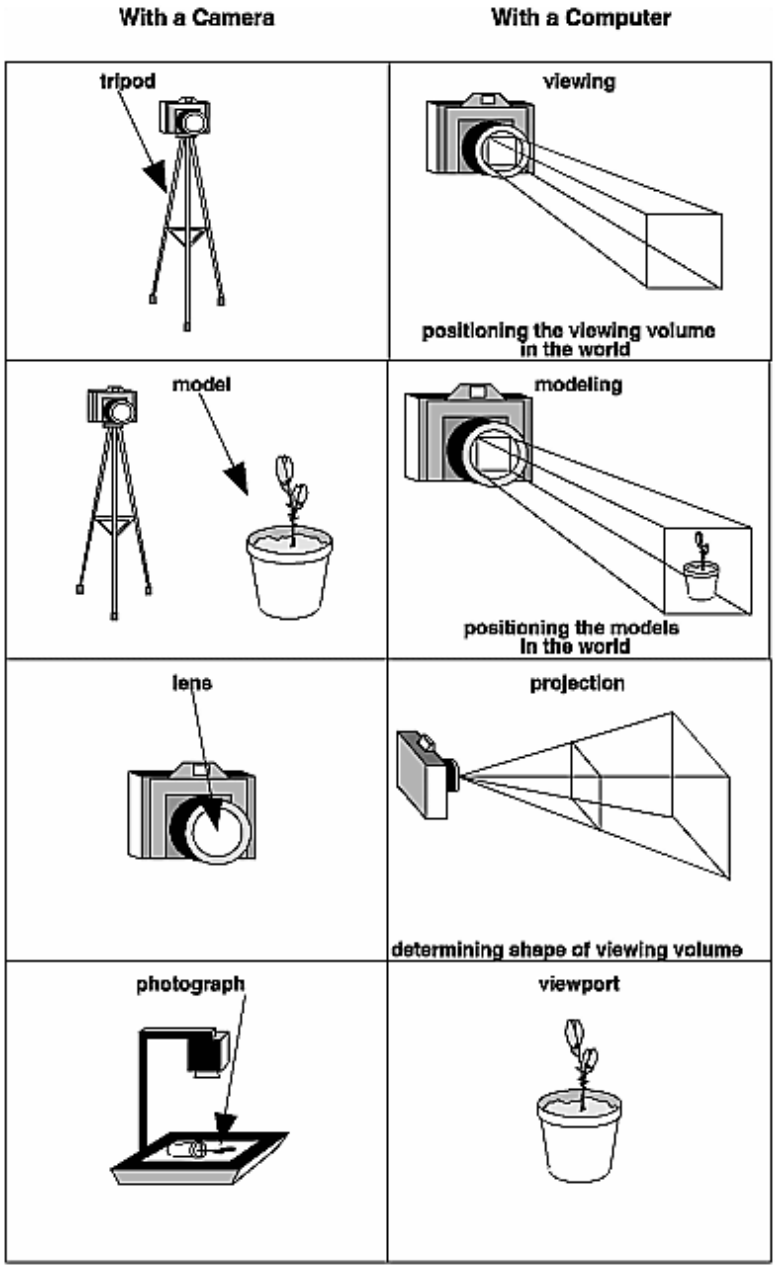
Contents

- OpenGL Basics
- Drawing Geometric Objects
- Viewing
- Color
- Lighting

OpenGL Viewing

- Read the Redbook Chapter 3

http://www.glprogramming.com/red/chapter03.html

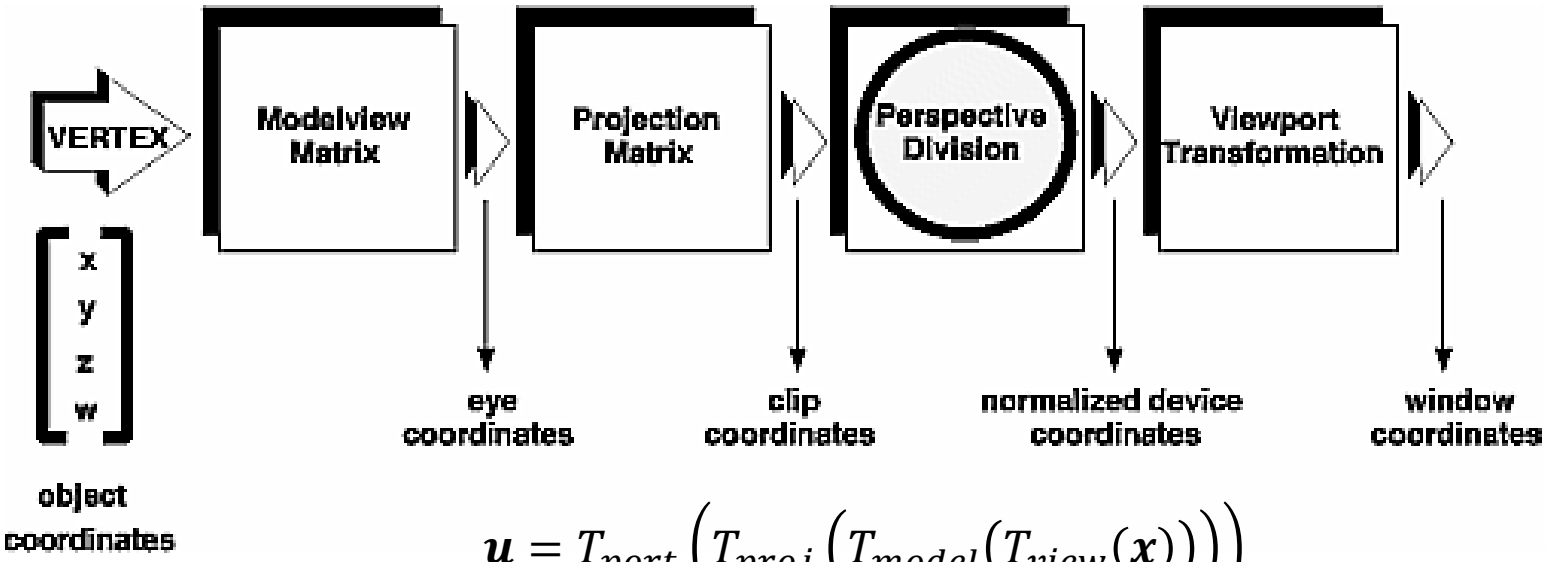


The camera analogy

Orders of these transformations in your program:

- The viewing, modeling transformations are from 3D to 3D;
- the projection transformations are from 3D to 2D;
- Viewport transformations are from 2D to 2D.
- They are all modeled using 4×4 transformation matrices (homogeneous representation).

The order of transformations in OpenGL execution:



$$\boldsymbol{u} = T_{port}\left(T_{proj}\left(T_{model}\left(T_{view}(\boldsymbol{x})\right)\right)\right)$$

1. Viewing + modeling transformations → stored together in the Modelview Matrix
2. Projection transformation → the Projection Matrix
   (viewing volume, clipping, get normalized device coordinates,
3. Viewport transformation
   (projected image transformed to window coordinates)

Viewing Transformation

- Will be set on a Modelview transformation matrix

1. Initialize it to the identity matrix using **glLoadIdentity()**

The composition of transformations in OpenGL:

- The newly added transformation commands will multiply the current matrix by the newly specified matrix and then set the result to be the current matrix.
- If you don't clear the current matrix (by loading it with the identity matrix), you continue to combine previous transformation matrices with the new one you supply.

Viewing Transformation (cont.)

After initialization:

- the viewing transformation can be specified with

  void **gluLookAt**(GLdouble eyex, GLdouble eyey,
  GLdouble eyez, GLdouble centerx, GLdouble centery,
  GLdouble centerz, GLdouble upx, GLdouble upy,
  GLdouble upz);

To indicate where the camera is placed, where it is aimed, and which way is up.

- By default (If **gluLookAt()** was not called), the setting is:

  gluLookat (0.0, 0.0, 0.0, 0.0, 0.0, -100.0, 0.0, 1.0, 0.0);

Model Transformation

- To position and orient the model
- For example, you can rotate, translate, or scale the model - or perform some combination of these operations.

- void **glTranslate**{fd}(TYPE x, TYPE y, TYPEz);
- void **glRotate**{fd}(TYPE angle, TYPE x, TYPE y, TYPE z);
- void **glScale**{fd}(TYPE x, TYPE y, TYPE z);
- void **glLoadMatrix**{fd}(const TYPE *m);
- void **glMultMatrix**{fd}(const TYPE *m);

Viewing + Modeling → ModelView

- In OpenGL, Viewing + Modeling transformations are combined together in a single ModelView matrix

- To manipulate the modeling/viewing transformation matrix, use:

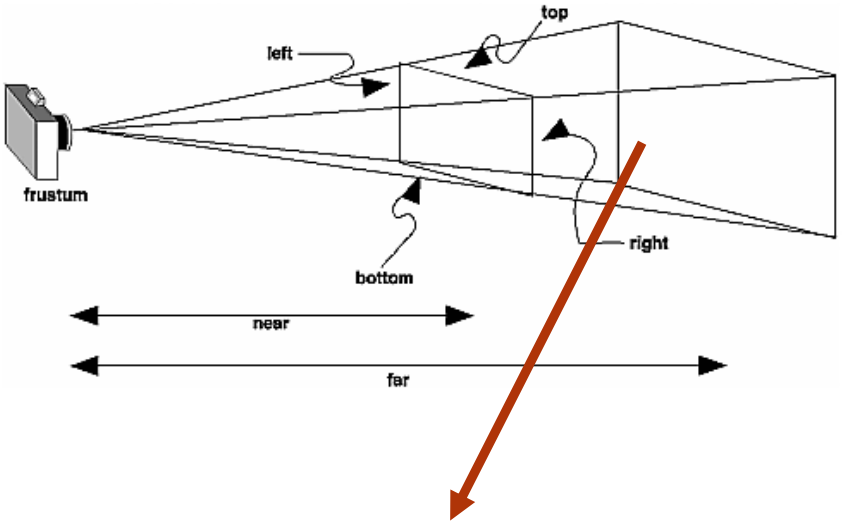
Void glMatrixMode(GL_MODELVIEW)

  - By default (if you didn't specify that you now are modifying ModelView matrix by GL_MODELVIEW), initially all the specified transformations are on ModelView matrix, until you change the status (by specifying GL_PROJECTION)
  - To make you program clear and easily readable, always specify this first

Projections

- After modeling and viewing transformations, the camera and objects are placed. Transformations are stored in the modelview matrix
- Next, define the desired projection matrix, which is also used to transform the vertices in your scene.
- Before you issue any of the transformation commands described in this section, remember to call

glMatrixMode(GL_PROJECTION);
glLoadIdentity();



- In OpenGL, this volume (view volume) will be normalized to a unit cube, after this Projection step.
- Everything outside the view volume will be ignored.

Projections

Generally:

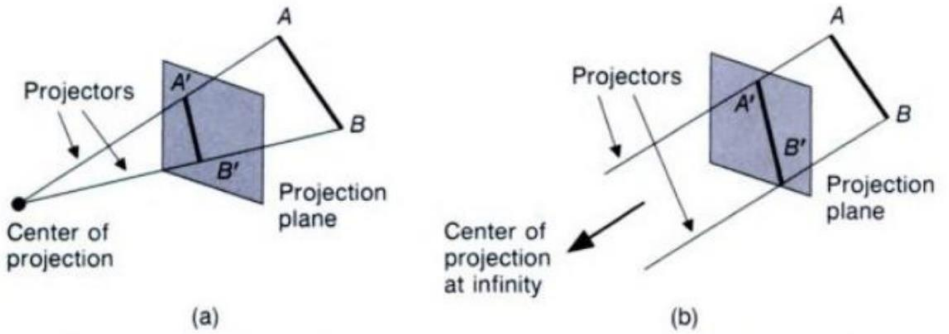
❑Projections transform points in a n-D coordinate system into points in a m-D coordinate system (m<n)

❑Computer Graphics has long been used for studying n-D objects by projecting them into lower dimensional (especially, 2D) space.

Noll, M. , "A Computer Technique for Displaying N-dimensional hyperobjects", CACM, 10(8), Aug. 1967, 469-473.

Here:

❑We focus on projections from 3D to 2D.



(a) Line AB and its perspective projection $A'B'$. (b) Line AB and its parallel projection $A'B'$. Projectors AA' and BB' are parallel.

❑Projection :

- ❑straight projection rays (**projectors**) emanating from a **center of projection**
- ❑Passing through each point of the object
- ❑Intersecting a **projection plane** to form the projection image

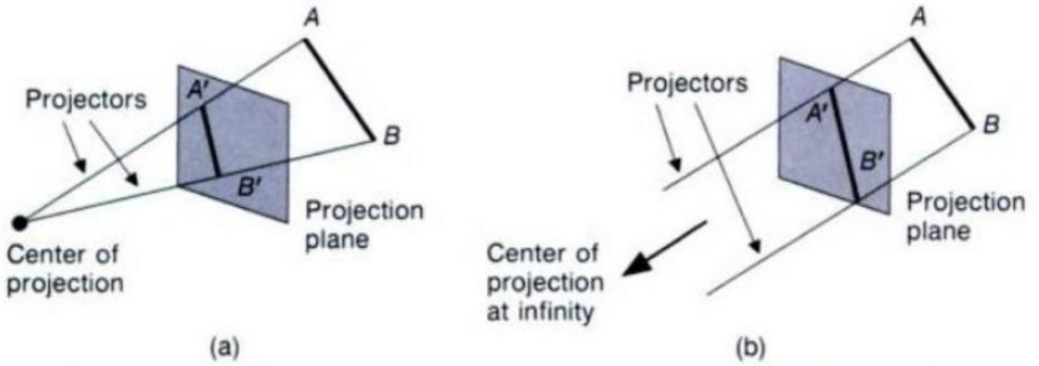
Classification of Projections

General Classification:

- We deal with planar geometric projections
- Non-planar projection: the projection plane is a curved surface (e.g. many cartographic projections)
- Non-geometric projection: the projection rays are curved (e.g. the Omnimax film)

Planar Geometric Projections:

- Parallel projection: projection center is infinitely far away
  - so that all projectors are parallel
  - We only need to specify direction of projection
- Perspective projection: projection center is finite distance away
  - Need to specify projection center



(a) Line AB and its perspective projection $A'B'$. (b) Line AB and its parallel projection $A'B'$. Projectors AA' and BB' are parallel.

Classification of Projections

General Classification:

Planar Geometric Projections:

- Perspective projection:
  - Visually : perspective foreshortening (the object size varies inversely with the distance from the projection center), similar to human visual system
  - Measurement:
    - not good for recording exact shape,
    - angles are preserved only on those faces of the object parallel to the projection plane,
    - parallel lines generally are not projected to be parallel
- Parallel projection:
  - Visually : less realistic
  - Measurement:
    - good for exact measurement,
    - parallel lines remain parallel,
    - angles only preserved on faces that are parallel to the projection plane

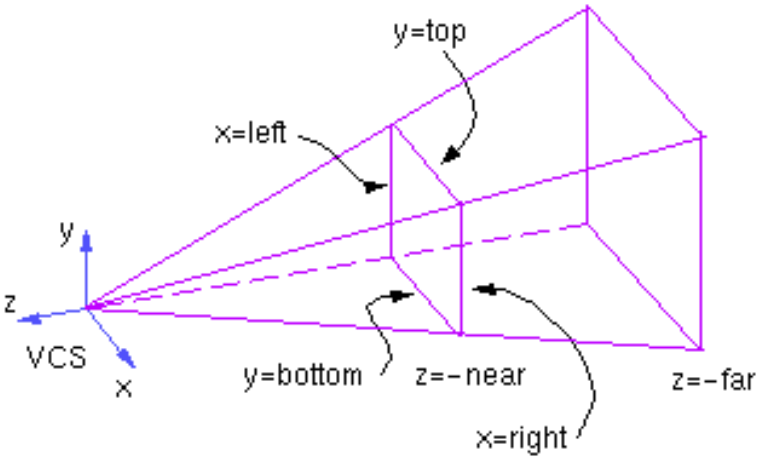
For more detailed discussions, check: Carlbom, I. and J. Paciorek, "Planar Geometric Projections and Viewing Transformations", Computing Surveys, 10(4), Dec. 1978, pp. 465-502.

Projection Transformations in OpenGL (cont.)

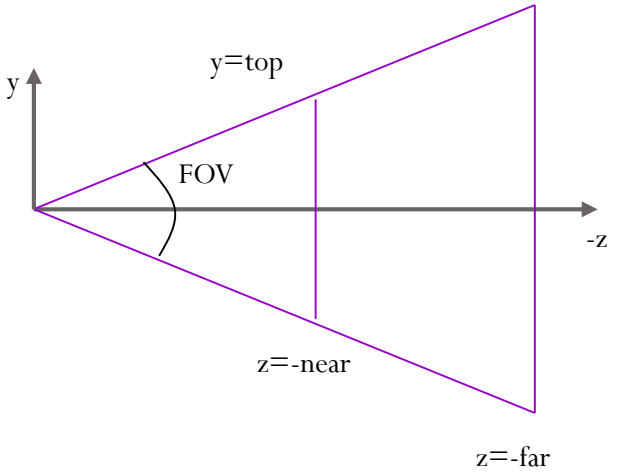
- **Perspective Projection**

  Two simple ways to specify the pyramid:

  - void glFrustum(double left, double right, double bottom, double top, double near, double far);

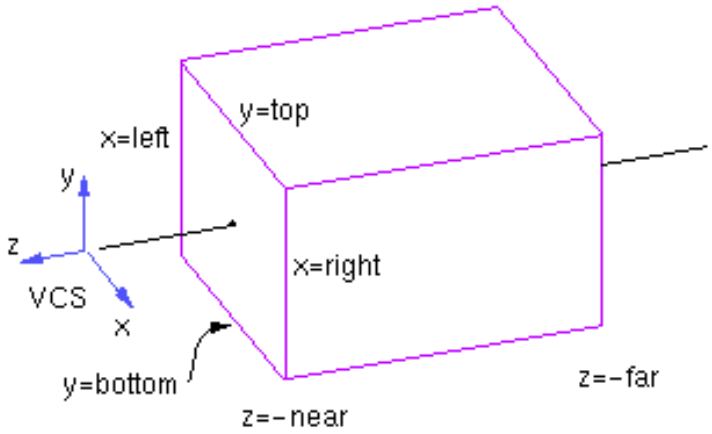


  - void gluPerspective(double fovy, double aspect, double zNear, double zFar);
    - fovy: field of view angle in y direction
    - aspect: aspect ratio = fov-size-x / fov-size-y
    - zNear, zFar: distance from the viewer to the near and far clipping planes, both are always positive



Projection Transformations in OpenGL (cont.)

- **Orthographic Projection**
  - void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble zNear, GLdouble zFar);
  - void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top);



Summary:
A Common Routine in Setting Projection Transformations:

- First, put:
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
- Then, put one of:
  glFrustrum, gluPerspective, or glOrtho

Ex4.cpp

Pay attention to the set of transformations and their orders to be applied.

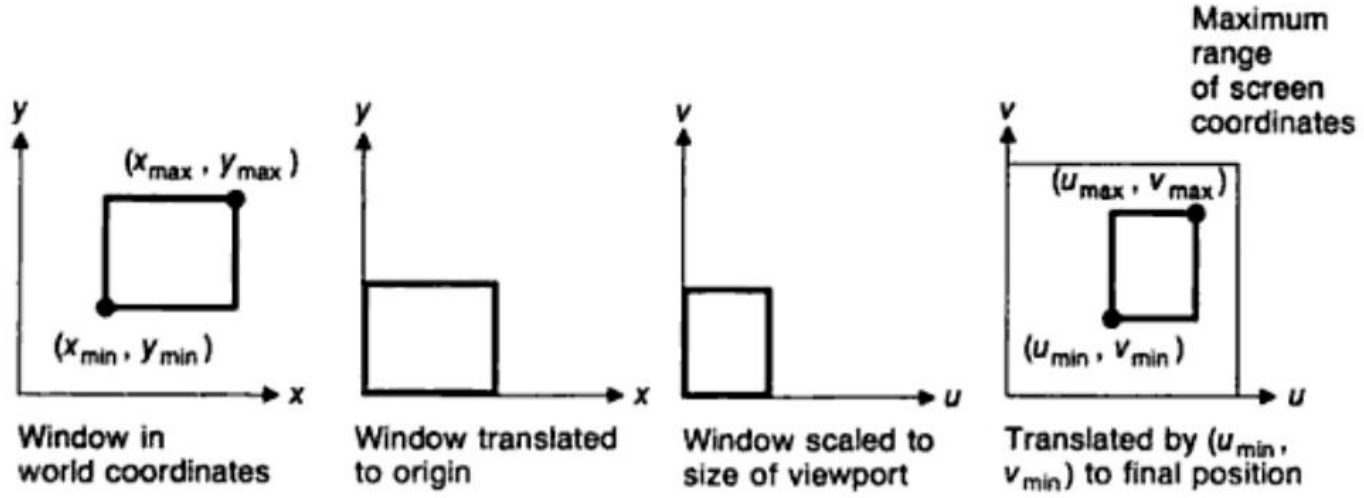
(1) the transformation you do will affect all the subsequent lines;

(2) If you want to do some transformations to only one object, you may need to “undo” the transformations after plotting this object.

(3) Instead of doing an inverse transformation to undo some transformation, you can use glPushMatrix() and glPopMatrix() to save a current configuration (the composed matrix) into a stack and load it.

2D Window-To-Viewport Transformation

- objects (2D) are now in the normalized device coordinate (NDC) system
- need to map them onto screen coordinates
- This steps solves:
  - Given a rectangular region in a 2D coordinates system (NDC)
  - A user specifies the corresponding rectangular region in screen coordinates (viewport)
  - → To find the 2D to 2D transformation



Sometimes, we want shape-preserving: make $s_u = s_v$

Viewport Transformation in OpenGL

- By default the viewport is set to the entire pixel rectangle of the window that's opened.
- You can use the **glViewport()** command to choose a smaller drawing region; for example, you can subdivide the window to create a split-screen effect for multiple views in the same window.
- **void glViewport(GLint x, GLint y, GLsizei width, GLsizei height)**;
  - Defines a pixel rectangle in the window into which the final image is mapped.
  - (x, y)→ lower-left corner of the viewport,
  - width and height → size of the viewport rectangle.
  - By default, the initial viewport values are (0, 0, winWidth, winHeight), where winWidth and winHeight are the size of the window.
- The aspect ratio of a viewport is usually set to equal the aspect ratio of the viewing volume. If the two ratios are different, the projected image will be distorted when mapped to the viewport.
- To do this, our application should detect window resize events and modify the viewport appropriately.

Example Programs (Ex5.cpp)

```
void display (void) {
    …
}

void reshape(int w, int h)
{
    glViewport (0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    gluPerspective(60, 1, 1,10);                          // try this viewport setting first
    //gluPerspective(60, (float)w / (float)h, 1, 10);     // then try this viewport setting
    glutPostRedisplay();
}

int main(int argc, char** argv) {
            …
}
```

Setting 1 will map a square to the entire window.
Setting 2 will keep the aspect ratio, by adjusting the camera (film) aspect ratio accordingly.

Example Programs (Ex6.cpp)

1. Check the print line in the reshape() function, and find out when will the reshape() function be triggered.
2. In the display callback function, we get to the ModelView mode, initialize the matrix to identity, then use GluLookAt to set up the camera.
3. To transform the object(s) to render, just call glTranslate/glRotate to apply the transformations.
4. If you have multiple objects that you want to apply different transformation. You can use glPushMatrix() and glPopMatrix() to save and restore previous transformation setting (ModelView Matrix).

Example Programs (Ex6.cpp)

1. Check the print line in the reshape() function, and find out when will the reshape() function be triggered.
2. In the display callback function, we get to the ModelView mode, initialize the matrix to identity, then use GluLookAt to set up the camera.
3. To transform the object(s) to render, just call glTranslate/glRotate to apply the transformations.
4. If you have multiple objects that you want to apply different transformation. You can use glPushMatrix() and glPopMatrix() to save and restore previous transformation setting (ModelView Matrix).

Example Programs (Ex7.cpp)

To rotate or translate objects differently:

1. From initial camera setting, right before rendering each object, save current matrix status, then call approparite glRotate/glTranslate
2. After redering each object, use glPopMatrix to restore camera configuration, then do the transformation for the next object…

Some other notes:

1. glShadeModel (GL_FLAT) : the polygon will be rendered using a constant color, while GL_SMOOTH is the other (default) option, where the polygon will be interpolated using colors of vertices
2. Here the Pentagon is rendered using 3 composed triangles.