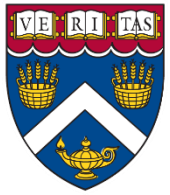


HARVARD
Extension School

CSCI E-74
Virtual and Augmented Reality for Simulation and Gaming

Fall term 2017

Gianluca De Novi, PhD

Lesson 3 – General Introduction to OpenGL APIs and TRS

HARVARD
UNIVERSITY

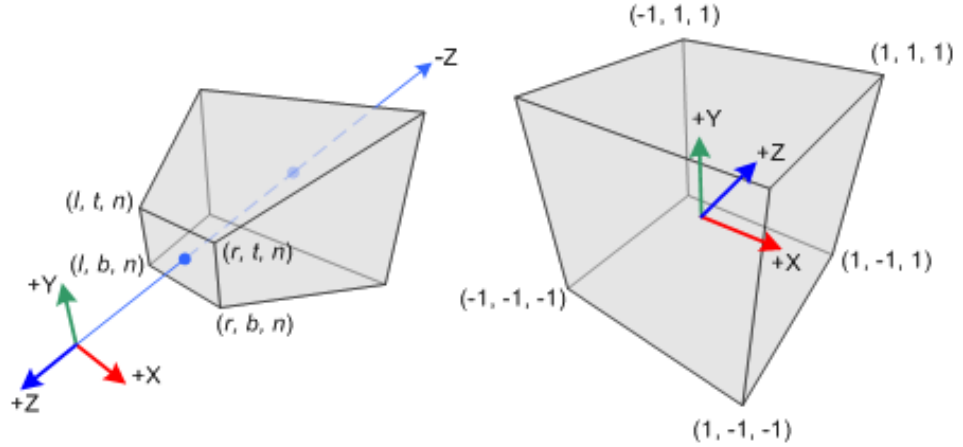
Perspective Simulation

Perspective simulation is achieved using the projection matrix GL_PROJECTION to map the 3D image into the 2D viewport (eye coordinates).

$$x \in [l, r]$$
$$y \in [b, t]$$
$$z \in [n, f]$$

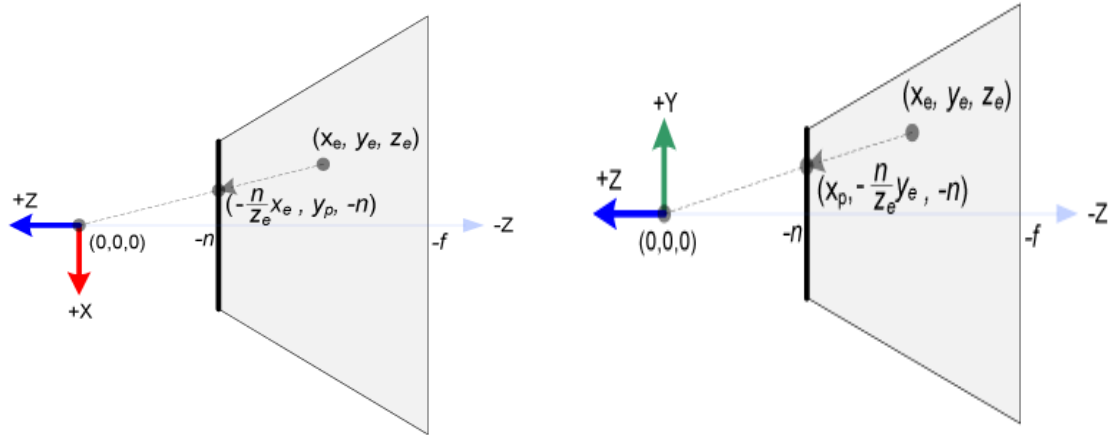
$$\frac{x_p}{x_e} = \frac{-n}{z_e} \Rightarrow x_p = \frac{-nx_e}{ze} = \frac{nx_e}{-ze}$$

$$\frac{y_p}{y_e} = \frac{-n}{z_e} \Rightarrow y_p = \frac{-ny_e}{ze} = \frac{ny_e}{-ze}$$



$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix}$$

GL_PROJECTION matrix



Matrix Mode

Specify which matrix is the current matrix

```
void glMatrixMode(Glenum mode);
```

GL_MODELVIEW Applies subsequent matrix operations to the modelview matrix stack.

GL_PROJECTION Applies subsequent matrix operations to the projection matrix stack.

GL_TEXTURE Applies subsequent matrix operations to the texture matrix stack.

GL_COLOR Applies subsequent matrix operations to the color matrix stack.

Identity Matrix

Load Identity matrix

```
void glLoadIdentity();
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Load Generic Matrix

Load Identity matrix

```
void glLoadmatrixd(const GLdouble * m);
void glLoadmatrixf(const GLfloat * m);
```

Example

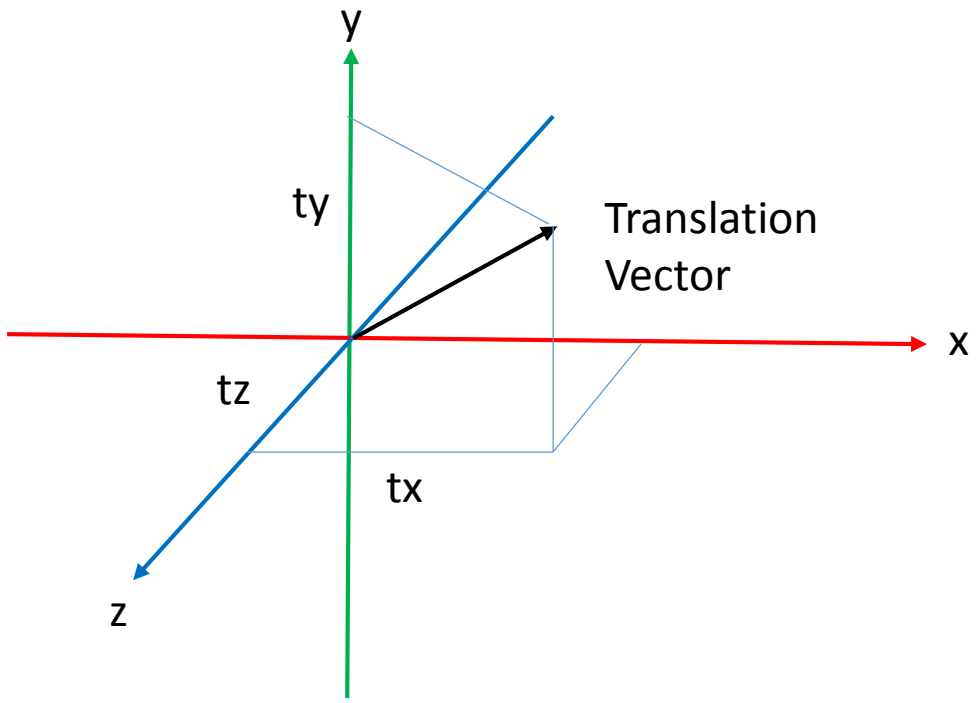
```
float m[16]={1.0f,0.0f,0.0f,0.0f,
             0.0f,1.0f,0.0f,0.0f,
             0.0f,0.0f,1.0f,0.0f,
             0.0f,0.0f,0.0f,1.0f}; //example of identity matrix
…
glLoadMatrixf(m);
…
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translation Matrix

```
void glTranslatef(GLfloat tx, GLfloat ty, GLfloat tz);

void glTranslated(GLdouble tx, GLdouble ty, GLdouble tz);
```

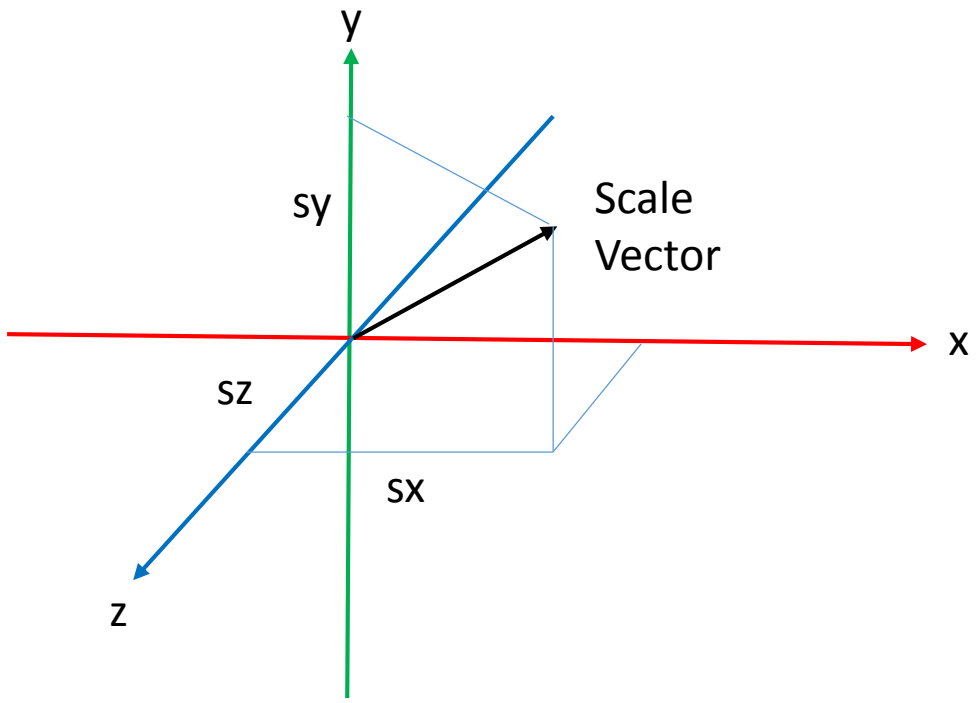


$$\begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scale Matrix

```
void glScalef(GLfloat sx, GLfloat sy, GLfloat sz);

void glScaled(GLdouble sx, GLdouble sy, GLdouble sz);
```



$$\begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation Matrix

```
void glRotatef(GLfloat angle, GLfloat rx, GLfloat ry, GLfloat rz);

void glRotated(GLdouble angle, GLdouble rx, GLdouble ry, GLdouble rz);
```

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

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

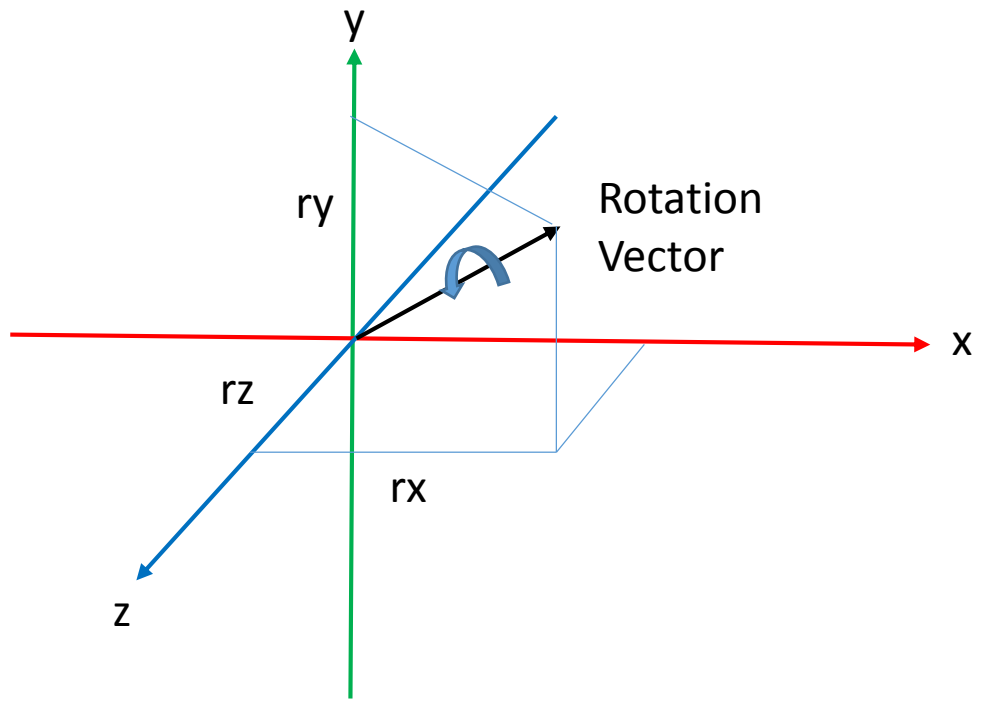
$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Combining

$$\begin{pmatrix} x^2(1-c)+c & xy(1-c)-zs & xz(1-c)+ys & 0 \\ xy(1-c)+zs & y^2(1-c)+c & yz(1-c)-xs & 0 \\ xz(1-c)-ys & yz(1-c)+xs & z^2(1-c)+c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where $c = \cos\theta, \quad s = \sin\theta$

Rotation around a generic vector matrix



TRS Complex

For a generic object

```
void BuildObjMatrix()
{
  glMatrixMode(GL_MODELVIEW);              // select the modelview mode transformation of the entities
  glLoadIdentity();                        // loads the identity matrix
  glTranslatef(obj.tx,obj.ty,obj.tz);      // translate the object to the desired position
  glRotatef    (obj.yaw  ,0.0f,1.0f,0.0f); // rotation around Y axis
  glRotatef    (obj.pitch,1.0f,0.0f,0.0f); // rotation around X axis
  glRotatef    (obj.roll ,0.0f,0.0f,1.0f); // rotation around Z axis
  glScalef     (obj.sx,obj.sy,obj.sz);     // Scale object
}
```

For the Camera

```
void BuildCameraMatrix()
{
  glMatrixMode(GL_MODELVIEW);               // select the modelview mode transformation of the entities
  glLoadIdentity();                         // loads the identity matrix
  glRotatef    (-pov.yaw  ,0.0f,1.0f,0.0f); // rotation around Y axis
  glRotatef    (-pov.pitch,1.0f,0.0f,0.0f); // rotation around X axis
  glRotatef    (-pov.roll ,0.0f,0.0f,1.0f); // rotation around Z axis
  glTranslatef(-pov.tx,-pov.ty,pov.tz);     // translate the camera to the desired position
}
```

For the camera, NO SCALE and OPPOSITE ORDER

TRS Complex

```
void BuildObjMatrix(…)
{
  glMatrixMode(GL_MODELVIEW);              // select the modelview mode transformation of the entities
  glLoadIdentity();                        // loads the identity matrix
  glTranslatef(obj.tx,obj.ty,obj.tz);      // translate the object to the desired position
  glRotatef    (obj.yaw  ,0.0f,1.0f,0.0f); // rotation around Y axis
  glRotatef    (obj.pitch,1.0f,0.0f,0.0f); // rotation around X axis
  glRotatef    (obj.roll ,0.0f,0.0f,1.0f); // rotation around Z axis
  glScalef     (obj.sx,obj.sy,obj.sz);     // Scale object
}
```

$$\begin{bmatrix} x_t \\ y_t \\ z_t \\ 1 \end{bmatrix} = \underbrace{\left(S \times R_r \times R_p \times R_y \times T\right)}_{\boldsymbol{Mo}} \times \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

TRS Complex

```
void BuildCameraMatrix()
{
  glMatrixMode(GL_MODELVIEW);              // select the modelview mode transformation of the entities
  glLoadIdentity();                        // loads the identity matrix
  glRotatef    (-pov.yaw  ,0.0f,1.0f,0.0f); // rotation around Y axis
  glRotatef    (-pov.pitch,1.0f,0.0f,0.0f); // rotation around X axis
  glRotatef    (-pov.roll ,0.0f,0.0f,1.0f); // rotation around Z axis
  glTranslatef(-pov.tx,-pov.ty,pov.tz);     // translate the camera to the desired position
}
```

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \underbrace{\left(R_y \times R_p \times R_r \times T\right)}_{\boldsymbol{Mc}} \times \begin{bmatrix} x_t \\ y_t \\ z_t \\ 1 \end{bmatrix}$$

3DGE Workflow

```
#define N_ENTITIES 100

int i;
TEntity  Entity[N_ENTITIES];
… load entities somewhere
do{
   … generate entities configuration somewhere
   BuildCameraMatrix();
   for(i=0;i< N_ENTITIES;i++)
   {
        BuildObjMatrix(Entity[i]);
   }

   CleanFrameBuffer();
   for(i=0;i< N_ENTITIES;i++)
   {
        DrawEntity(Entity[i]);
   }
   FlipBuffer();

}While(CheckExitEvent());
```

Number or entities in our simulation
Just a simple variable to use as counter
Array of entities

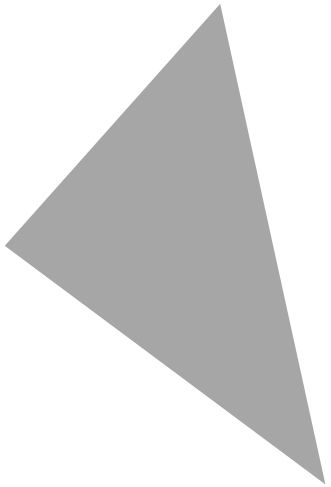
Simulation Loop

Generates the geometry information

Draws the final scene

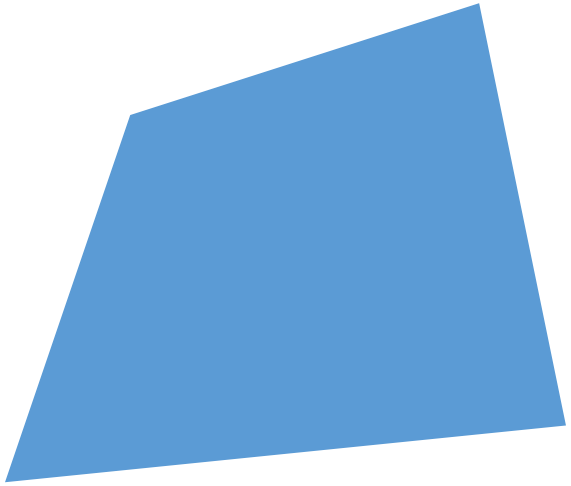
Triangle Quadrilateral Drawing

```
glBegin(GL_TRIANGLES);
   glVertex3f(v1.x,v1.y,v1.z);
   glVertex3f(v2.x,v2.y,v2.z);
   glVertex3f(v3.x,v3.y,v3.z);
glEnd();
```



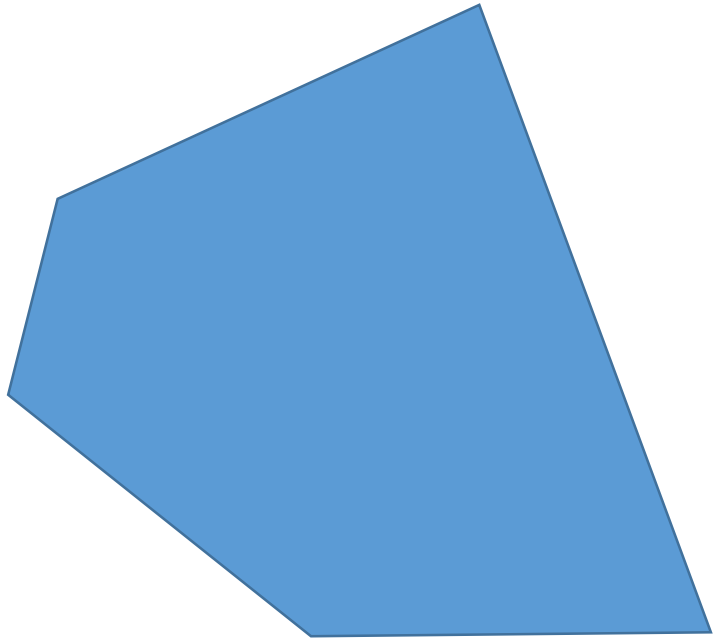
f=floating point
single precision

```
glBegin(GL_QUADS);
   glVertex3f(v1.x,v1.y,v1.z);
   glVertex3f(v2.x,v2.y,v2.z);
   glVertex3f(v3.x,v3.y,v3.z);
   glVertex3f(v4.x,v4.y,v4.z);
glEnd();
```



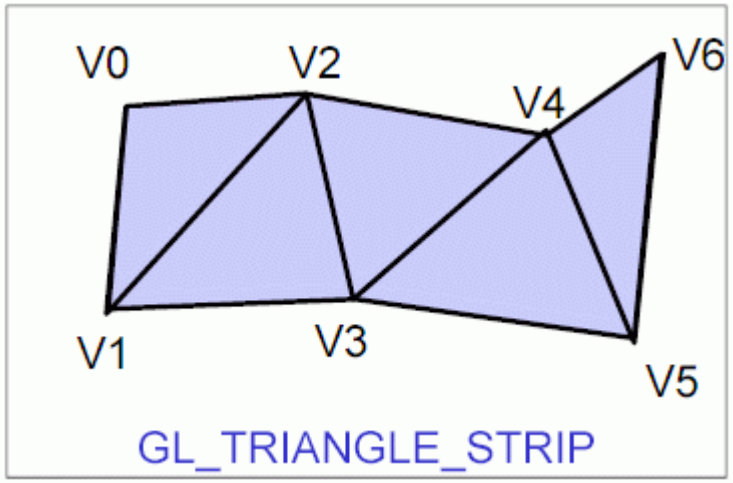
Polygon

```
glBegin(GL_POLYGON);
   glVertex3f(v1.x,v1.y,v1.z);
   glVertex3f(v2.x,v2.y,v2.z);
   glVertex3f(v3.x,v3.y,v3.z);
   glVertex3f(v4.x,v4.y,v4.z);
   glVertex3f(v5.x,v5.y,v5.z);
glEnd();
```



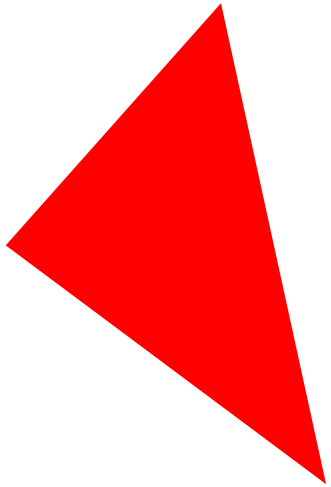
GL-LINES

```
glColor4f(1.0f,0.0f,0.0f,1.0f);
glBegin(GL_TRIANGLE_TRIP);
   glVertex3f(v0.x,v0.y,v0.z);
   glVertex3f(v1.x,v1.y,v1.z);
   glVertex3f(v2.x,v2.y,v2.z);
   glVertex3f(v3.x,v3.y,v3.z);
   glVertex3f(v4.x,v4.y,v4.z);
   glVertex3f(v5.x,v5.y,v5.z);
   glVertex3f(v6.x,v6.y,v6.z);
glEnd();
```



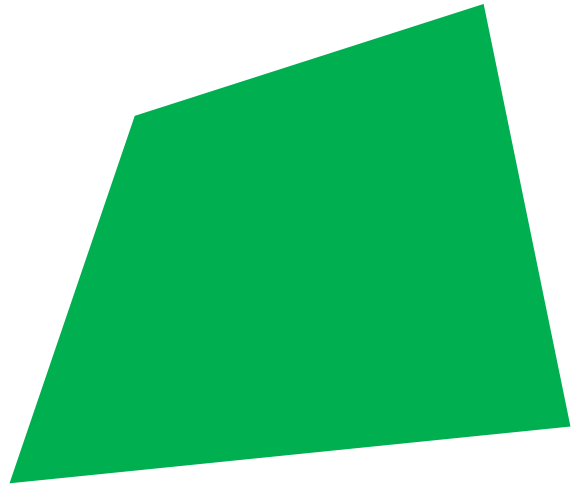
Color

```
glColor4f(1.0f,0.0f,0.0f,1.0f);
glBegin(GL_TRIANGLE);
   glVertex3f(v1.x,v1.y,v1.z);
   glVertex3f(v2.x,v2.y,v2.z);
   glVertex3f(v3.x,v3.y,v3.z);
glEnd();
```



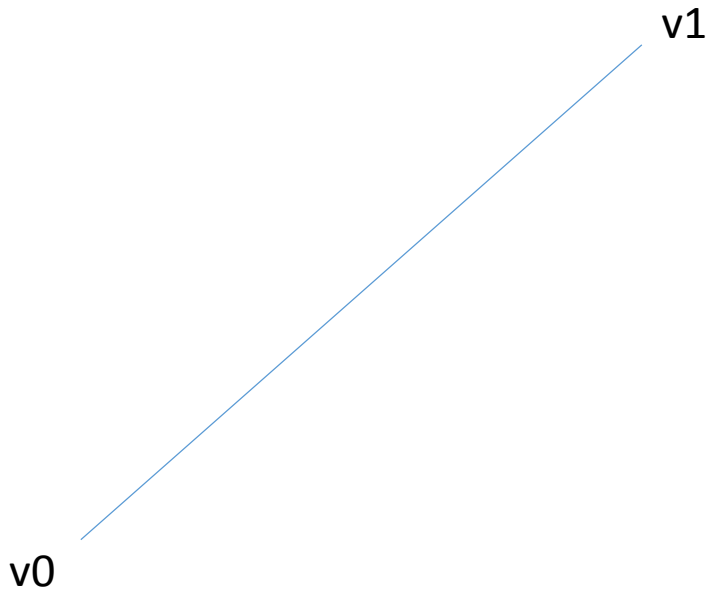
$r \in \boldsymbol{R}\ [0,1]$
$g \in \boldsymbol{R}\ [0,1]$
$b \in \boldsymbol{R}\ [0,1]$
$a \in \boldsymbol{R}\ [0,1]$

```
glColor4f(0.0f,1.0f,0.0f,1.0f);
glBegin(GL_QUAD);
   glVertex3f(v1.x,v1.y,v1.z);
   glVertex3f(v2.x,v2.y,v2.z);
   glVertex3f(v3.x,v3.y,v3.z);
   glVertex3f(v4.x,v4.y,v4.z);
glEnd();
```



Triangle Strip

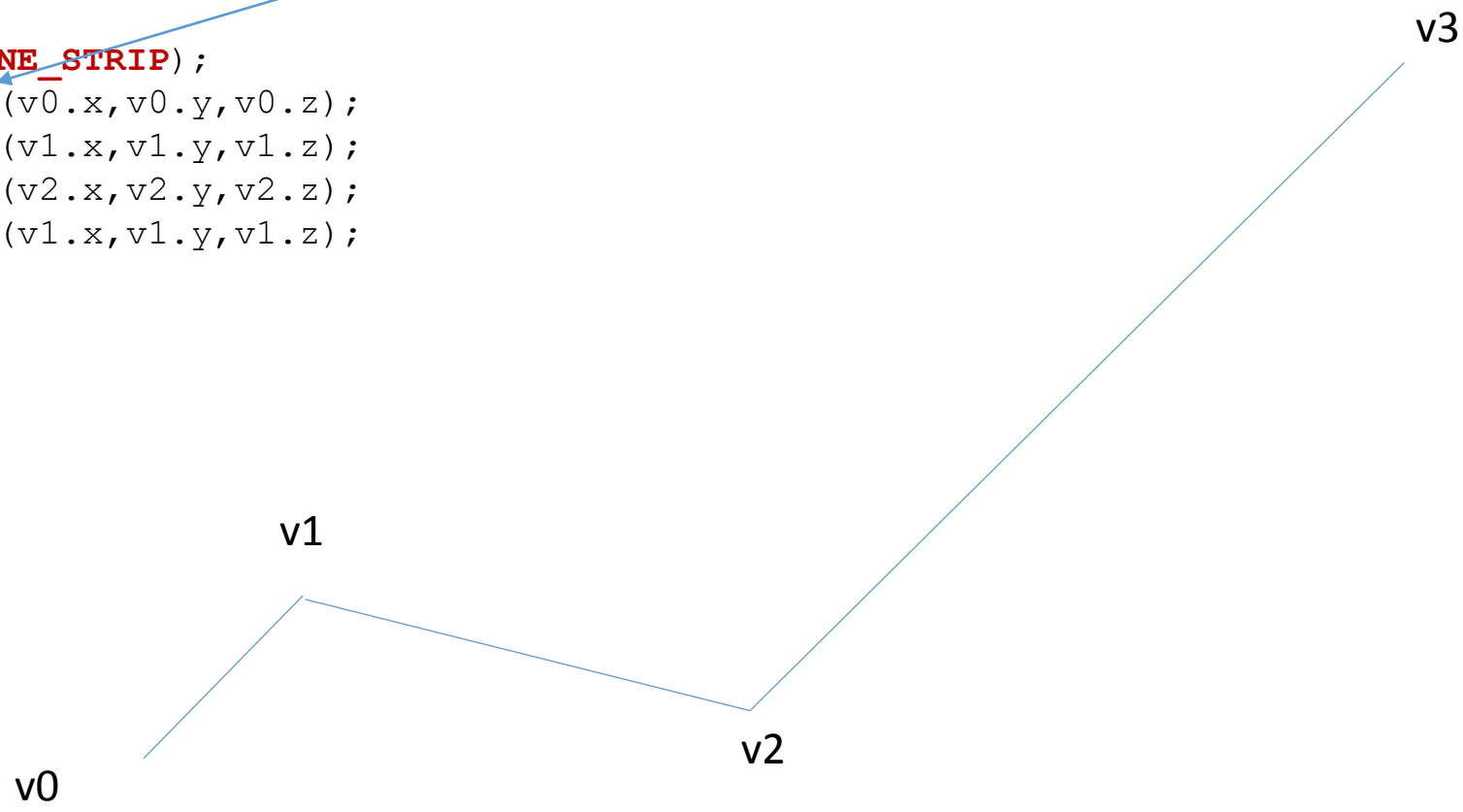
```
glBegin(GL_LINES);
   glVertex3f(v0.x,v0.y,v0.z);
   glVertex3f(v1.x,v1.y,v1.z);
glEnd();
```



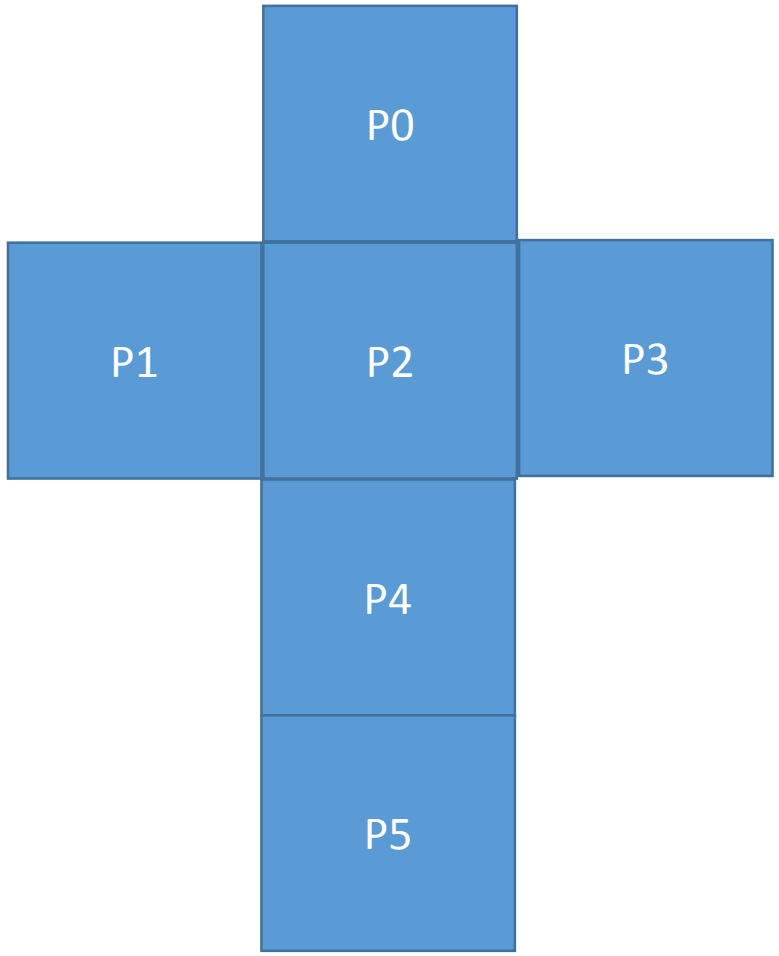
Triangle Strip

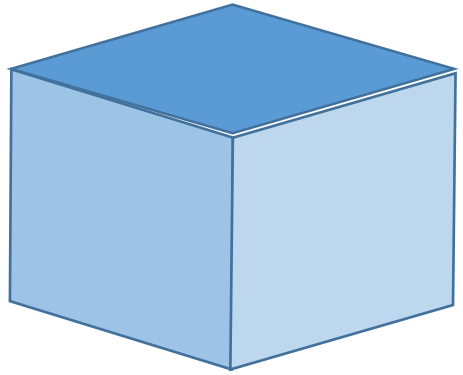
Double precision

```
glBegin(GL_LINE_STRIP);
   glVertex3d(v0.x,v0.y,v0.z);
   glVertex3d(v1.x,v1.y,v1.z);
   glVertex3d(v2.x,v2.y,v2.z);
   glVertex3d(v1.x,v1.y,v1.z);
glEnd();
```



Hidden faces





I cannot see all the faces at the same time!

Polygon Normal

$$\bar{V}_1 = (V_{1x}, V_{1y}, V_{1z}) = (v1_x - v0_x, v1_y - v0_y, v1_z - v0_z) \in \boldsymbol{R}^3$$

$$\bar{V}_2 = (V_{2x}, V_{2y}, V_{2z}) = (v2_x - v0_x, v2_y - v0_y, v2_z - v0_z) \in \boldsymbol{R}^3$$

$$V_1 = |\bar{V}_1| = \sqrt{(v1_x - v0_x)^2 + (v1_y - v0_y)^2 + (v1_z - v0_z)^2} \in \boldsymbol{R}$$

$$V_2 = |\bar{V}_2| = \sqrt{(v2_x - v0_x)^2 + (v2_y - v0_y)^2 + (v2_z - v0_z)^2} \in \boldsymbol{R}$$

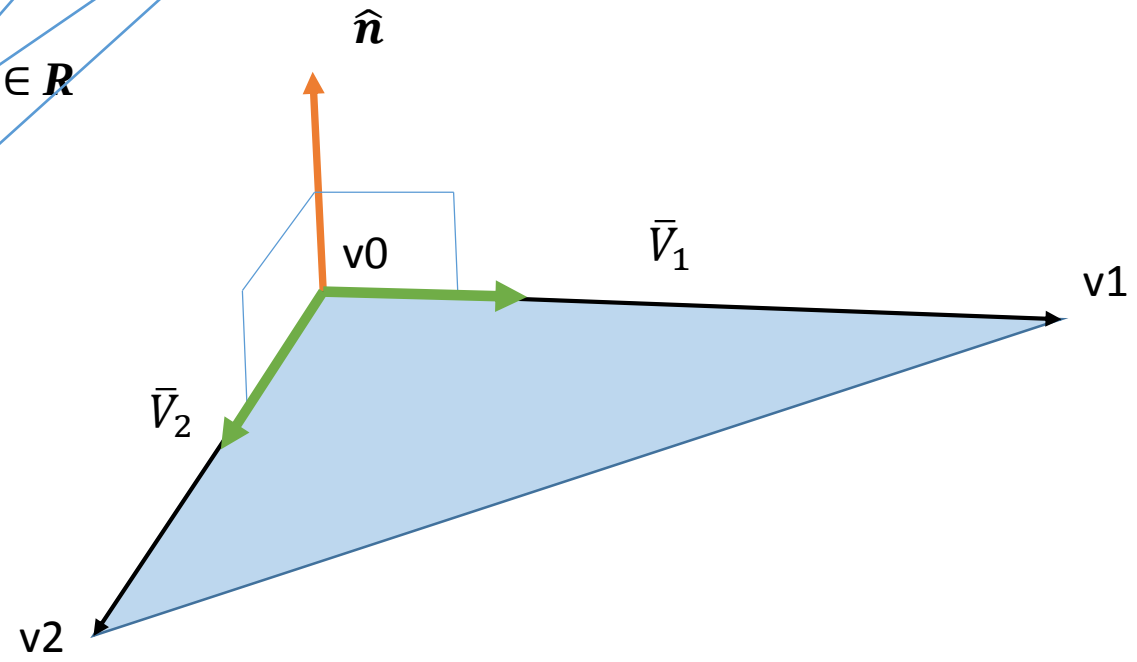
$$\widehat{V_1} = \|\bar{V}_1\| = \left(\frac{v1_x - v0_x}{|\boldsymbol{V}_1|}, \frac{v1_y - v0_y}{|\boldsymbol{V}_1|}, \frac{v1_z - v0_z}{|\boldsymbol{V}_1|}\right) \in \boldsymbol{R}^3$$

$$\widehat{V_2} = \|\bar{V}_2\| = \left(\frac{v2_x - v0_x}{|\boldsymbol{V}_1|}, \frac{v2_y - v0_y}{|\boldsymbol{V}_1|}, \frac{v2_z - v0_z}{|\boldsymbol{V}_1|}\right) \in \boldsymbol{R}^3$$

$$\hat{\boldsymbol{n}} = \|\bar{V}_1\| \times \|\bar{V}_2\| = \widehat{V_1} \times \widehat{V_2}$$

Represent the length of $\bar{V}_1$ and $\bar{V}_2$

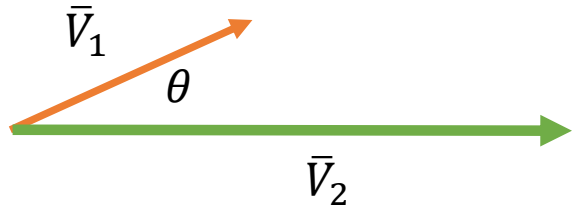
Represent the direction of $\bar{V}_1$ and $\bar{V}_2$



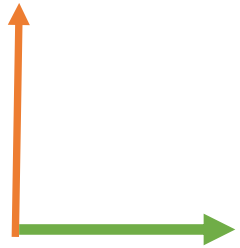
Scalar Product

$$\bar{V}_1 \cdot \bar{V}_2 = V_1 V_2 \cos\theta$$

$$\bar{V}_1 \cdot \bar{V}_2 = \left(V_{1x}V_{2x} + V_{1y}V_{2y} + V_{1z}V_{2z}\right) \in \boldsymbol{R}$$



1



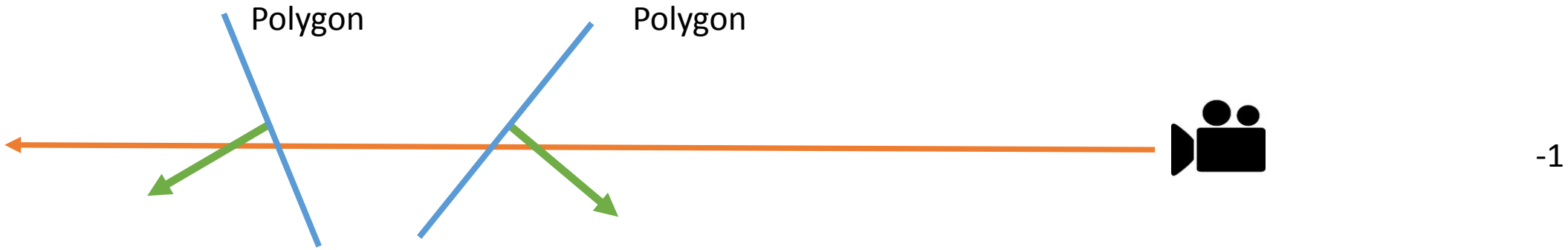
0

-1

Scalar Product

$$\bar{V}_1 \cdot \bar{V}_2 = V_1 V_2 \cos\theta$$

$$\bar{V}_1 \cdot \bar{V}_2 = \left(V_{1x}V_{2x} + V_{1y}V_{2y} + V_{1z}V_{2z}\right) \in \boldsymbol{R}$$

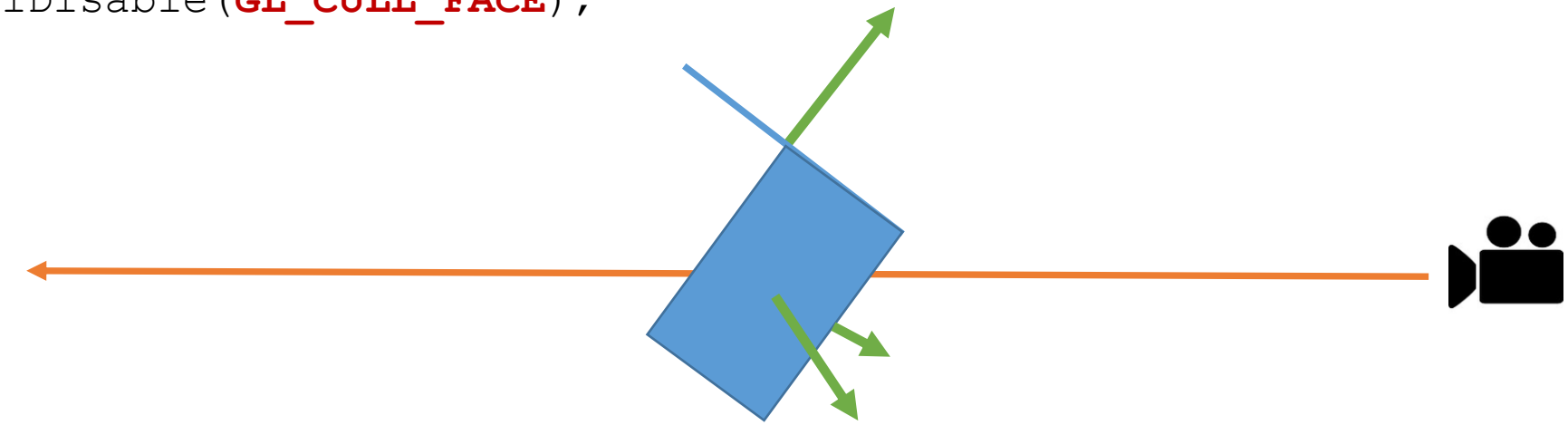


$$if\ \bar{V}_1 \cdot \bar{V}_2 \leq 0\ \ then\ draw\ primitive$$
$$else\ \ discard$$

GL_CULL_FACE

```
glEnable(GL_CULL_FACE);
glDisable(GL_CULL_FACE);
```

Only 3 faces



$$if\ \bar{V}_1 \cdot \bar{V}_2 \leq 0\quad then\ draw\ primitive$$
$$else\qquad\qquad discard$$

Polygon Normals

Specifies the normal versor

```
glNormal3f(GLfloat nx, Glfloat ny, Glfloat nz);
```

Example

```
glBegin(GL_POLYGON);
   glNormal3f(n.x, n.y, n.z);
   glVertex3f(v1.x, v1.y, v1.z);
   glVertex3f(v2.x, v2.y, v2.z);
   glVertex3f(v3.x, v3.y, v3.z);
glEnd();
```