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

Fall term 2017

Gianluca De Novi, PhD

Lesson 4 – Basic Architecture Implementation

HARVARD
UNIVERSITY

Perspective Correction

```
void glHint( GLenum target, GLenum mode);
```

Certain aspects of GL behavior, when there is room for interpretation, can be controlled with hints. A hint is specified with two arguments. *target* is a symbolic constant indicating the behavior to be controlled, and *mode* is another symbolic constant indicating the desired behavior. The initial value for each *target* is GL_DONT_CARE. *mode* can be one of the following:

GL_FASTEST The most efficient option should be chosen.
GL_NICEST The most correct, or highest quality, option should be chosen.
GL_DONT_CARE No preference.

Though the implementation aspects that can be hinted are well defined, the interpretation of the hints depends on the implementation. The hint aspects that can be specified with *target*, along with suggested semantics, are as follows:

GL_FOG_HINT Indicates the accuracy of fog calculation. If per-pixel fog calculation is not efficiently supported by the GL implementation, hinting GL_DONT_CARE or GL_FASTEST can result in per-vertex calculation of fog effects.
GL_FRAGMENT_SHADER_DERIVATIVE_HINT Indicates the accuracy of the derivative calculation for the GL shading language fragment processing built-in functions: dFdx, dFdy, and fwidth.
GL_GENERATE_MIPMAP_HINT Indicates the quality of filtering when generating mipmap images.
GL_LINE_SMOOTH_HINT Indicates the sampling quality of antialiased lines. If a larger filter function is applied, hinting GL_NICEST can result in more pixel fragments being generated during rasterization.
GL_PERSPECTIVE_CORRECTION_HINT Indicates the quality of color, texture coordinate, and fog coordinate interpolation. If perspective-corrected parameter interpolation is not efficiently supported by the GL implementation,
hinting GL_DONT_CARE or GL_FASTEST can result in simple linear interpolation of colors and/or texture coordinates.
GL_POINT_SMOOTH_HINT Indicates the sampling quality of antialiased points. If a larger filter function is applied, hinting GL_NICEST can result in more pixel fragments being generated during rasterization.
GL_POLYGON_SMOOTH_HINT Indicates the sampling quality of antialiased polygons. Hinting GL_NICEST can result in more pixel fragments being generated during rasterization, if a larger filter function is applied.
GL_TEXTURE_COMPRESSION_HINT Indicates the quality and performance of the compressing texture images. Hinting GL_FASTEST indicates that texture images should be compressed as quickly as possible, while GL_NICEST indicates that texture images
should be compressed with as little image quality loss as possible.GL_NICEST should be selected if the texture is to be retrieved by glGetCompressedTexImage for reuse.

Automatic Vectors Normalization

Enable the automatic normalization

```
void glEnable(GL_NORMALIZE);
```

If enabled, normal vectors are normalized to unit length after transformation and before lighting.

This method is generally less efficient than GL_RESCALE_NORMAL.

See glNormal and glNormalPointer.

Shade Model

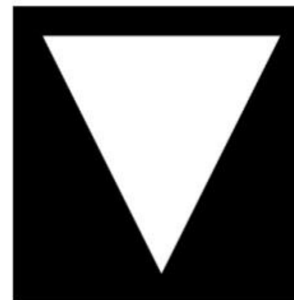
Selects the shading mode

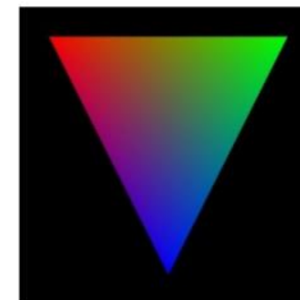
```
void glShadeModel(GLenum mode);
```

`mode` Specifies a symbolic value representing a shading technique. Accepted values are GL_FLAT and GL_SMOOTH. The initial value is GL_SMOOTH.

Simplest Fragment Shading

- Flat color shading
  - glShadeModel(GL_FLAT)
- Interpolated color shading
  - glShadeModel(GL_SMOOTH)





HARVARD Extension School

HARVARD UNIVERSITY

Rasterization Mode

select a polygon rasterization mode

```
void glPolygonMode(GLenum face, GLenum mode);
```

`face` Specifies the polygons that mode applies to. Must be GL_FRONT_AND_BACK for front- and back-facing polygons.

`mode` Specifies how polygons will be rasterized. Accepted values are GL_POINT, GL_LINE, and GL_FILL. The initial value is GL_FILL for both front- and back-facing polygons.

Trigonometry Notes

Trigonometric functions in C

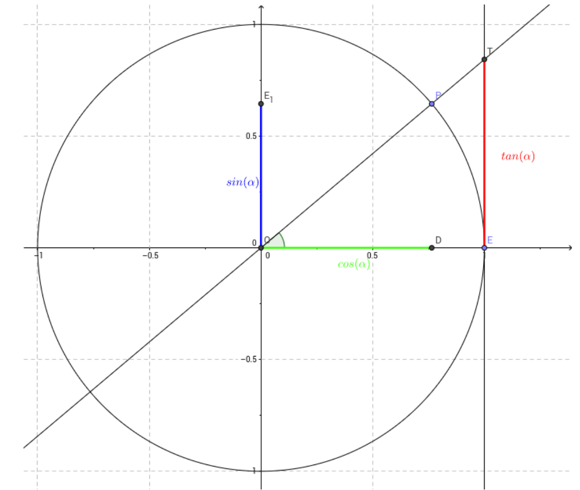
```
float sin(float angle);
float cos(float angle);
float tan(float angle);
float asin(float d);
float acos(float d);
float atan(float d);
```

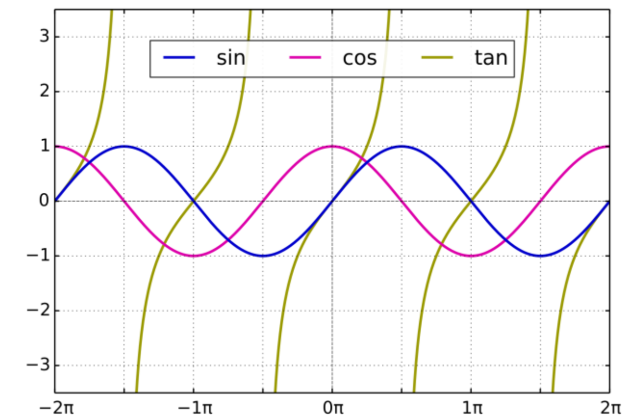
Angle is expressed in radians!

Degrees
{Conversion} = {Chart}

Degrees	Radians	Degrees	Radians
0°	0	210°	7 π/6
30°	π/6	225°	5 π/4
45°	π/4	240°	4 π/3
60°	π/3	270°	3 π/2
90°	π/2	300°	5 π/3
120°	2 π/3	315°	7 π/4
135°	3 π/4	330°	11 π/6
150°	5 π/6	360°	2 π
180°	π		

www.SwiftTips.com





Trigonometry Notes

Degree to Radians

1° = π/180° = 0.005555556π = 0.01745329252 rad

Then AR=AD* 0.01745329252

Example

```
#define D2R 0.017453292519943295769236907684
```

```
A = sin(angle*D2R);
```

Degrees

{Conversion} = {Chart}

Degrees	Radians	Degrees	Radians
0°	0	210°	7 π/6
30°	π/6	225°	5 π/4
45°	π/4	240°	4 π/3
60°	π/3	270°	3 π/2
90°	π/2	300°	5 π/3
120°	2 π/3	315°	7 π/4
135°	3 π/4	330°	11 π/6
150°	5 π/6	360°	2 π
180°	π		

www.SwiftTips.com

Trigonometry Notes

Radians to Degrees

1 rad = 180°/π = 57.295779513°

Then AD=AR* 57.295779513°

Example

```
#define R2D 57.295779513082320876798154817014

angle = asin(1)*R2D;
```

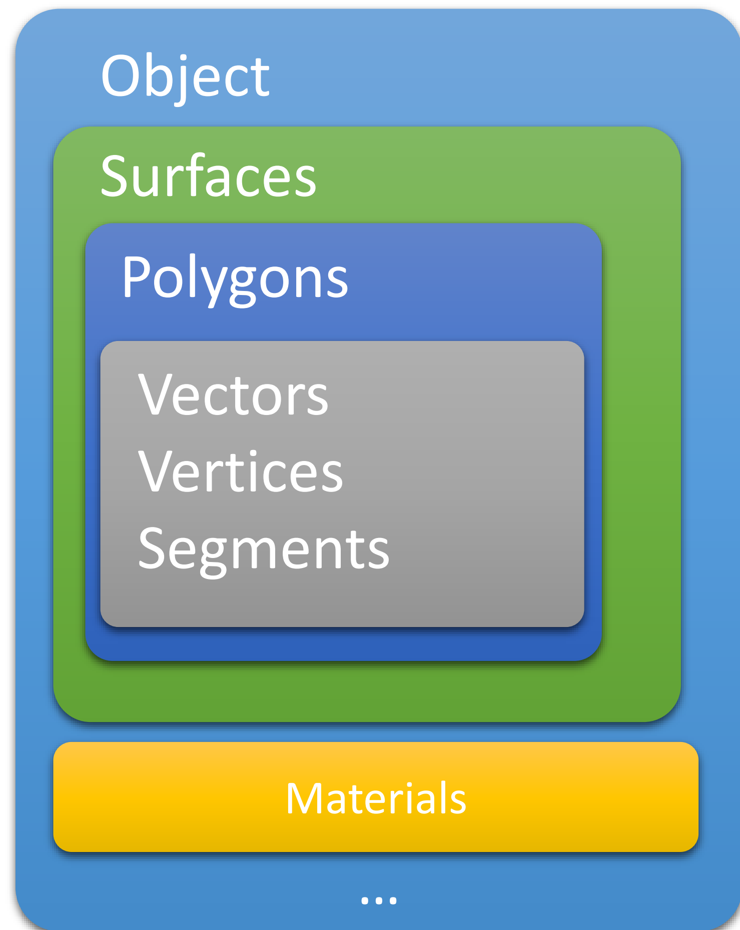
Degrees
{Conversion} = {Chart}

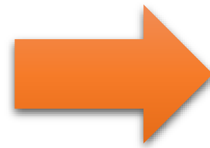
Degrees	Radians	Degrees	Radians
0°	0	210°	7 π/6
30°	π/6	225°	5 π/4
45°	π/4	240°	4 π/3
60°	π/3	270°	3 π/2
90°	π/2	300°	5 π/3
120°	2 π/3	315°	7 π/4
135°	3 π/4	330°	11 π/6
150°	5 π/6	360°	2 π
180°	π		

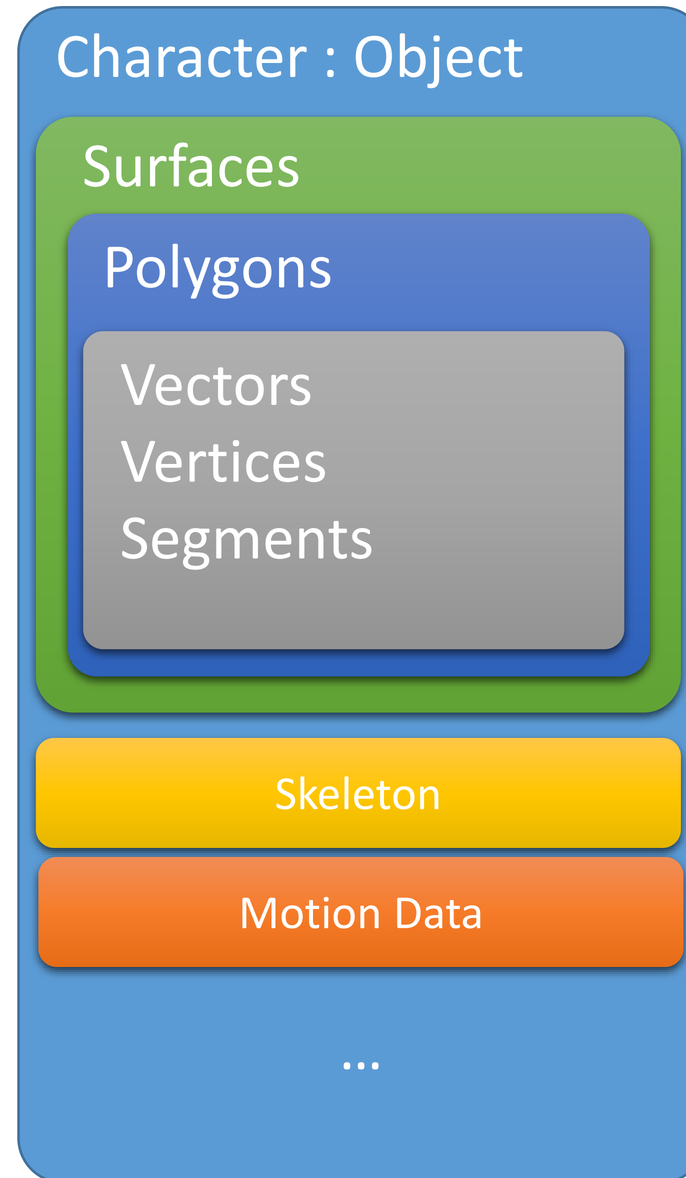
Data Structures in a 3D Engine

- **Vertices/Vectors**
- Segments
- **Matrices**
- **Polygons**
- **Surfaces**
- **Materials**
- **Objects (Generic)**
- Super categories of Objects
- Lights
- Motion Data
- Lattices
- Skeletons
- Textures
- Particles
- …

Data Hierarchy







Definition of a Vector Class

```
Class TVector
{
 public:
  float x,y,z;
  TVector() {x=y=z=0;}
  void Clear();
  void vector(float xx,float yy,float zz);
  int operator ==  (TVector &v);
  int operator !=  (TVector &v);
  TVector operator - (TVector &v);
  TVector operator + (TVector &v);
  float  operator * (TVector &v);
  TVector operator ^ (TVector &v);
  TVector operator / (TVector &v);
  TVector &operator -= (TVector &v);
  TVector &operator += (TVector &v);
  TVector &operator *= (TVector &v);
  TVector operator - (float &v);
  TVector operator + (float &v);
  TVector operator * (float &v);
  TVector operator / (float &v);
  TVector &operator -= (float &v);
  TVector &operator += (float &v);
  TVector &operator *= (float &v);
  TVector &operator /= (float &v);
  void Normalize();
  void Normalize(float MaxModule);
  float Module();
  void Scale(float s);
  void Reflect(TVector n);
  void Refract(TVector n, float RefIndex);
};
Typedef Vector Vertex;
```

Initialization Methods

Logic Operators

Vector - Vector Operators

Vector - Scalar Operators

Vector module operations

Reflection and Refraction

Implementation Examples

```
int TVector::operator ==  (Vector &v)
{
  if(v.x==x && v.y==y && v.z==z) return 1;
  return 0;
}
```

```
TVector TVector::operator +  (TVector &v)
{
   TVector t;
   t.x=v.x+x;
   t.y=v.y+y;
   t.z=v.z+z;
   return t;
}
```

```
float TVector::operator *  (TVector &v)
{
   return v.x*x + v.y*y + v.z*z;
}
```

```
TVector TVector::operator ^  (TVector &v)
{
   TVector t;
   t.x=(y*v.z-z*v.y);
   t.y=(z*v.x-x*v.z);
   t.z=(x*v.y-y*v.x);
   return t;
}
```

```
void TVector::Normalize()
{
  float mod=(float)sqrt((float)x*x+y*y+z*z);
  if(mod!=0.0f)
  {
    mod = 1.0f/mod;
    x=(float)x*mod;
    y=(float)y*mod;
    z=(float)z*mod;
  }
}
```

```
void TVector::Reflect(TVector n)
{
   TVector t =  * this;
   n.Scale(2.0f*(t*n));
   *this = t-n;
}
```

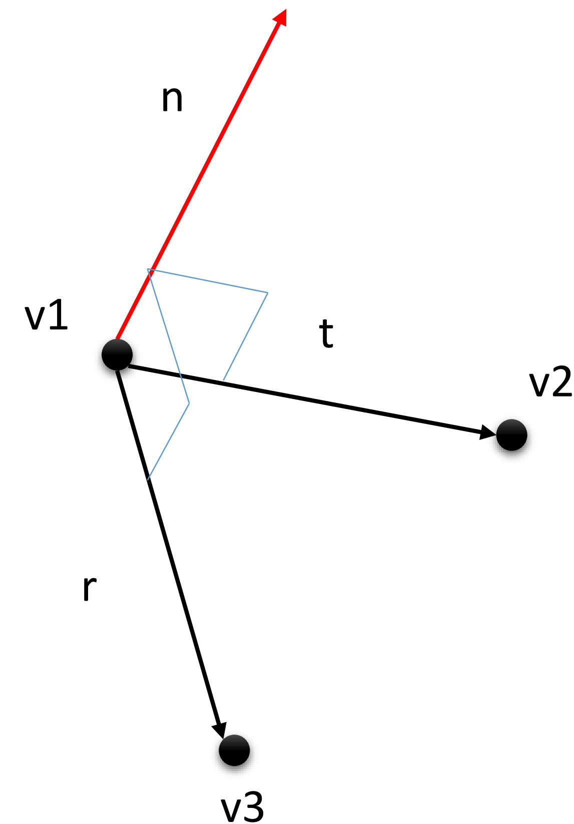
```
void TVector::Refract(TVector n,float RefIndex)
{
   n.Scale(-RefIndex);
   Vector t =  *this;
   t.Scale(1.0f-RefIndex);
   *this = t+n;
}
```

Assignment: Build your own version of this library

Normal calculation

```
TVertex v1,v2,v3;
TVector t,r,n;

//initialization of v1,v2,v3 somewhere in the code
…
t = v2-v1;
r = v3-v1;
n = t^r;
```



Now we know how to calculate normals

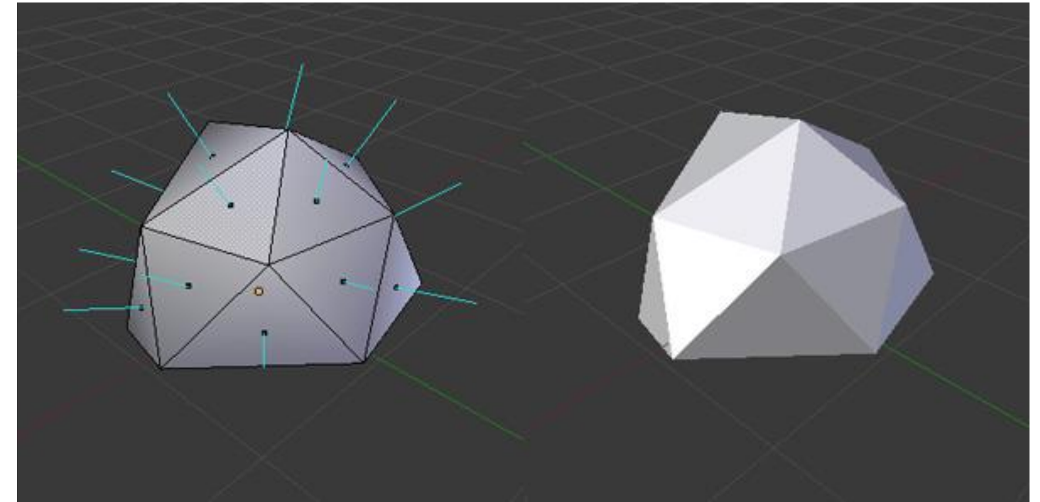
```
for(i=0;i<PolyListLen;i++)
{
   TVector t,r,n;

   t=Vlist[p[i].v2]-Vlist[p[i].v1];
   r=Vlist[p[i].v3]-Vlist[p[i].v1];

   p[i].Normal = t^r;
   p[i].Normal.Normalize();
}

//When I draw then…

glBegin(GL_TRIANGLES);
   glNormal3f(p[i].n.x, p[i].n.y, p[i].n.z);
   glVertex3f(Vlist[p[i].v1].x, Vlist[p[i].v1].y, Vlist[p[i].v1].z);
   glVertex3f(Vlist[p[i].v2].x, Vlist[p[i].v2].y, Vlist[p[i].v2].z);
   glVertex3f(Vlist[p[i].v3].x, Vlist[p[i].v3].y, Vlist[p[i].v3].z);
glEnd();
```



HARVARD Extension School

HARVARD UNIVERSITY

Definition of a Pose Class

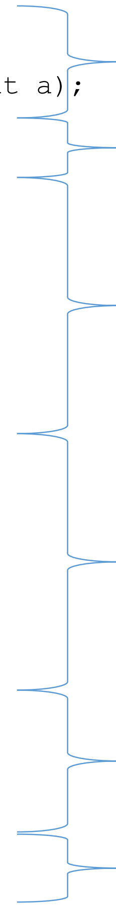
```
Class TPose:TVector
{
 public:
  float roll,pitch,yaw;
  TPose() {x=y=z=roll=pitch=yaw=0;}
};
```

Derived from the class TVector

Redefines the class constructor

Definition of a Color Class

```
Class TColor
{
 public:
  float r,g,b,a;
  TColor() {r=g=b=a=0;}
  void Clear();
  void color(float r,float g,float b,float a);
  int operator ==  (Color &v);
  int operator !=  (TColor &v);
  TColor operator - (TColor &v);
  TColor operator + (TColor &v);
  float  operator * (TColor &v);
  TColor operator ^ (TColor &v);
  TColor operator / (TColor &v);
  TColor &operator -= (TColor &v);
  TColor &operator += (TColor &v);
  TColor &operator *= (TColor &v);
  TColor operator - (float &v);
  TColor operator + (float &v);
  TColor operator * (float &v);
  TColor operator / (float &v);
  TColor &operator -= (float &v);
  TColor &operator += (float &v);
  TColor &operator *= (float &v);
  TColor &operator /= (float &v);
  void Normalize();
  void Normalize(float MaxModule);
  float Module();
  void Scale(float s);
};
```



Initialization Methods

Logic Operators

Vectorial Operators

Vector - Scalar Operators

Vector module operations

Reflection and Refraction

Definition of a Matrix 4x4 Class

```
class TMatrix4x4
{
 public:
    float m[16];

    TMatrix4x4() {Clear();}
    void  Clear();
    void  LoadIdentity();
    void  Transpose();
    void  Invert();
    void  Mul(TMatrix4x4 * M);
    void  Scalar(float s);
    float Determinant();
    …
    Vector     operator *   (Vector &v);
};
```

```
void TMatrix4x4::LoadIdentity()
{
   m[ 1]=m[ 2]=m[ 3]=m[ 4]=
   m[ 6]=m[ 7]=m[ 8]=m[ 9]=
   m[11]=m[12]=m[13]=m[14]=0.0f;
   m[ 0]=m[ 5]=m[10]=m[15]=1.0f;
}
```

```
void TMatrix4x4::Transpose()
{
   float a;
   a=m[ 1]; m[ 1]=m[ 4]; m[ 4]=a;
   a=m[ 2]; m[ 2]=m[ 8]; m[ 8]=a;
   a=m[ 3]; m[ 3]=m[12]; m[12]=a;
   a=m[ 6]; m[ 6]=m[ 9]; m[ 9]=a;
   a=m[ 7]; m[ 7]=m[13]; m[13]=a;
   a=m[11]; m[11]=m[14]; m[14]=a;
}
```

```
void TMatrix4x4::Mul(TMatrix4x4 * M)
{
  TMatrix4x4 r;
  r.m[ 0] = m[ 0]*M->m[ 0] + m[ 1]*M->m[ 4] + m[ 2]*M->m[ 8] + m[ 3]*M->m[12];
  r.m[ 1] = m[ 0]*M->m[ 1] + m[ 1]*M->m[ 5] + m[ 2]*M->m[ 9] + m[ 3]*M->m[13];
  r.m[ 2] = m[ 0]*M->m[ 2] + m[ 1]*M->m[ 6] + m[ 2]*M->m[10] + m[ 3]*M->m[14];
  r.m[ 3] = m[ 0]*M->m[ 3] + m[ 1]*M->m[ 7] + m[ 2]*M->m[11] + m[ 3]*M->m[15];

  r.m[ 4] = m[ 4]*M->m[ 0] + m[ 5]*M->m[ 4] + m[ 6]*M->m[ 8] + m[ 7]*M->m[12];
  r.m[ 5] = m[ 4]*M->m[ 1] + m[ 5]*M->m[ 5] + m[ 6]*M->m[ 9] + m[ 7]*M->m[13];
  r.m[ 6] = m[ 4]*M->m[ 2] + m[ 5]*M->m[ 6] + m[ 6]*M->m[10] + m[ 7]*M->m[14];
  r.m[ 7] = m[ 4]*M->m[ 3] + m[ 5]*M->m[ 7] + m[ 6]*M->m[11] + m[ 7]*M->m[15];

  r.m[ 8] = m[ 8]*M->m[ 0] + m[ 9]*M->m[ 4] + m[10]*M->m[ 8] + m[11]*M->m[12];
  r.m[ 9] = m[ 8]*M->m[ 1] + m[ 9]*M->m[ 5] + m[10]*M->m[ 9] + m[11]*M->m[13];
  r.m[10] = m[ 8]*M->m[ 2] + m[ 9]*M->m[ 6] + m[10]*M->m[10] + m[11]*M->m[14];
  r.m[11] = m[ 8]*M->m[ 3] + m[ 9]*M->m[ 7] + m[10]*M->m[11] + m[11]*M->m[15];

  r.m[12] = m[12]*M->m[ 0] + m[13]*M->m[ 4] + m[14]*M->m[ 8] + m[15]*M->m[12];
  r.m[13] = m[12]*M->m[ 1] + m[13]*M->m[ 5] + m[14]*M->m[ 9] + m[15]*M->m[13];
  r.m[14] = m[12]*M->m[ 2] + m[13]*M->m[ 6] + m[14]*M->m[10] + m[15]*M->m[14];
  r.m[15] = m[12]*M->m[ 3] + m[13]*M->m[ 7] + m[14]*M->m[11] + m[15]*M->m[15];
*this = r;
}
```

```
void TMatrix4x4::Clear()
{
   m[ 0]=m[ 1]=m[ 2]=m[ 3]=
   m[ 4]=m[ 5]=m[ 6]=m[ 7]=
   m[ 8]=m[ 9]=m[10]=m[11]=
   m[12]=m[13]=m[14]=m[15]=0.0f;
}
```

Assignment: Build your own version of this library

HARVARD Extension School

HARVARD UNIVERSITY

Polygon Class

```
class TPolygon
{
 public:
    int      v1,v2,v3;      //indices to the 3 vertices
    int      Visible;       //visibility flag
    TVector  Normal;        //Normal vector
    TColor   Color;         //color
    void     CalcNormal();  //calculate Normal
    void     Flip();        //flip the normal orientation
    void     Draw();        //draw
};
```

```
void  TPolygon::CalcNormal()
{
   Vector t,r,n;
   t = Vlist[v2]-Vlist[v1];
   r = Vlist[v3]-Vlist[v1];
   Normal = t^r;
   Normal.Normalize();
}
```

```
void  TPolygon::Flip()
{
   int a = v2;
   v2 = v3;
   v3 = v2;
   CalcNormal();
}
```

```
void  TPolygon::Draw()
{
   if(Visible)
   {
     glBegin(GL_TRIANGLES);
      glNormal3f(Normal.x, Normal.y, Normal.z);
      glVertex3f(Vlist[v1].x, Vlist[v1].y, Vlist[v1].z);
      glVertex3f(Vlist[v2].x, Vlist[v2].y, Vlist[v2].z);
      glVertex3f(Vlist[v3].x, Vlist[v3].y, Vlist[v3].z);
     glEnd();
   }
}
```

For example…   NB: This model doesn't consider the vertex optimization

Surface Class

```
class TSurface
{
 public:
    TVertex       VList[…];    //Vertices on the surface
    unsigned int  VListLen;    //Number of Vertices
    TPolygon      Plist[…];    //Polygons on the surface
    unsigned int  PListLen;    //Number of Polygons on the surface
    TColor        Color;       //Surface color
    int           Visible;     //Visibility Flag
    void          CalcNormals(); //calculate Normal
    void          Flip();        //flip the normal orientation
    void          Draw();        //draw
};
```

```
void  TSurface::CalcNormal()
{
  int i;
   for(i=0;i<PListLen;i++)
     Plist[i].CalcNormal();
}
```

```
void  TSurface::Flip()
{
   int i;
   for(i=0;i<PListLen;i++)
     Plist[i].Flip();
}
```

```
void  TSurface::Draw()
{
   if(Visible)
   {
      int i;
      glColor4f(Color.r,Color.g,Color.b,Color.a);
      for(i=0;i<PListLen;i++)
        Plist[i].Draw();
   }
}
```

For example… NB: This model doesn’t consider the vertex optimization

Object Class

```
class TObject
{
 public:
    TPose          Pose;              //Object Pose
    Tvector        Scale;             //Object Scale
    TMatrix4x4     TRS;               //Model View Matrix
    TSurface       SList[…];          //Surfaces of the Object
    unsigned int SListLen;            //Number of Surfaces
    int            Visible;           //Visibility Flag
    void           CalcNormals();     //calculate Normal
    void           BuildjMatrix();    //Builds the TRS Matrix
    void           Draw();            //draw
};
```

```
void  TObject::CalcNormal()
{
  int i;
   for(i=0;i<SListLen;i++)
     Slist[i].CalcNormal();
}
```

```
void  TObject::Draw()
{
   if(Visible)
   {
      int i;
      for(i=0;i<SListLen;i++)
               Slist[i].Draw();
   }
}
```

```
void  TObject:: BuildMatrix()
{
   glLoadIdentity();                            // loads the identity matrix
   glTranslatef(Pose.x,Pose.y,Pose.z);          // translate the object to the desired position
   glRotatef   (Pose.yaw  ,0.0f,1.0f,0.0f);     // rotation around Y axis
   glRotatef   (Pose.pitch,1.0f,0.0f,0.0f);     // rotation around X axis
   glRotatef   (Pose.roll ,0.0f,0.0f,1.0f);     // rotation around Z axis
   glScalef    (Scale.x,Scale.y,Scale.z);       // Scale object
   glGetFloatv( GL_MODELVIEW_MATRIX , TRS);
}
```

For example…   NB: This model doesn’t consider the vertex optimization

HARVARD Extension School

HARVARD UNIVERSITY

Camera Class

```
class TCamera
{
 public:
    TPose        POV;                 //Object Pose
    TMatrix4x4   TR;                  //TR Matrix
    void         BuildMatrix();       //Builds the TRS Matrix
};
```

```
void  TObject:: BuildMatrix()
{
   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
   glGetFloatv( GL_MODELVIEW_MATRIX , TR);
}
```

For example… NB: This model doesn't consider the vertex optimization

Scene Class

```
class TScene
{
 public:
    Tcamera       Camera;
    TObject       OList[…];          //List of objects
    unsigned int  OListLen;          //Number of objects
    int           Visible;           //Visibility Flag
    void          InitScene();       //Initialize Scene Geometry
    void          BuildSceneMatrices();//builds the scene matrices
    void          Draw();            //draw scene
};
```

```
void  TScene::InitScene()
{
  int i;
   for(i=0;i<OListLen;i++)
     Slist[i].CalcNormal();
}
```

```
void TScene:: BuildSceneMatrices()
{
   Camera.BuildMatrix()
   int i;
   for(i=0;i<OListLen;i++)
   OList[i].BuildMatrix();
}
```

```
void TScene::Draw()
{
   if(Visible)
   {
      int i;
      for(i=0;i<OListLen;i++)
      {
         glLoadMatrixf(Camera.TR);    //Load the Camera Matrix
         glMultMatrixf(Olist[i].TRS); //Multiplies for the Obj Matrix
         OList[i].Draw();
      }
   }
}
```

For example…   NB: This model doesn't consider the vertex optimization

EngineVR Class

```
class TEngineVR
{
 public:
    Tscene       SCList[…];     //Available Scenes
    unsigned int SCListLen;     //Number of Scenes
    unsigned int CurrentScene;  //Selected Scene
    void         InitScene();   //Initializes selected Scene
    void         RenderScene();//draw scene
};
```

```
void  TEngineVR::InitScene()
{
   SClist[CurrentScene].CalcNormal();
}
```

```
void TEngineVR::RenderScene()
{
  SCList[CurrentScene].Draw();
}
```

For example… NB: This model doesn't consider the vertex optimization

Data Structure

TEngineVR

TScene

TObject	TObject		TObject
Vectors Vertices Segments	Vectors Vertices Segments	…	Vectors Vertices Segments
Materials	Materials		Materials
Cameras	Motion Data	…	Lights

Data Structures in a 3D Engine

- **Vertices/Vectors**
- Segments
- **Matrices**
- **Polygons**
- **Surfaces**
- **Materials**
- **Objects (Generic)**
- Super categories of Objects
- Lights
- Motion Data
- Lattices
- Skeletons
- Textures
- Particles
- …