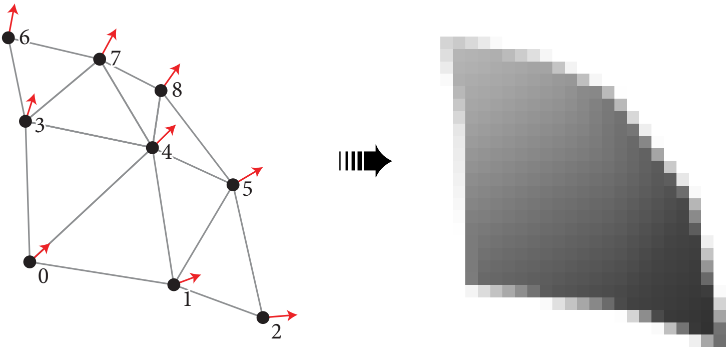
CSC 4356 / ME 4573 Interactive Computer Graphics

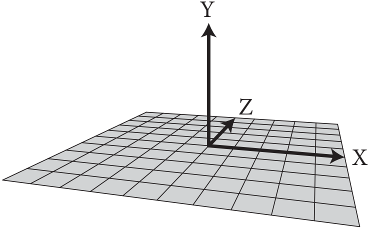
Basic Pipeline: Geometry, Transformation & Lighting



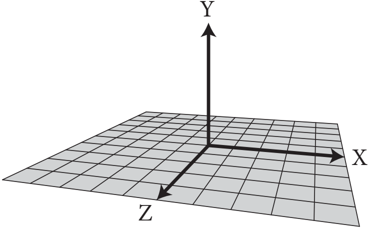
The 3D pipeline takes us from
geometry definition to rendered pixels.

3D coordinates

Assume X points to the right and Y points up.

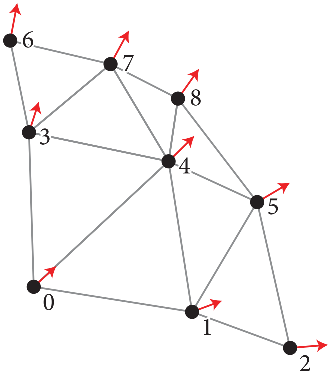


Left-handed: Z points *in*.

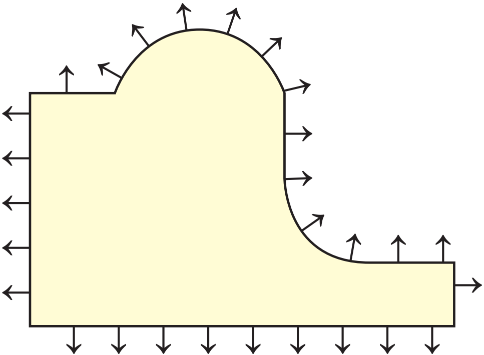


Right-handed: Z points *out*.

Each vertex has several attributes.

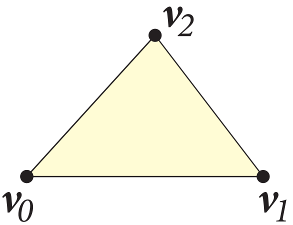


- 3D Position vector
- 3D Normal vector
- RGBA Color
- Texture coordinate(s)
- …*and more, depending.*

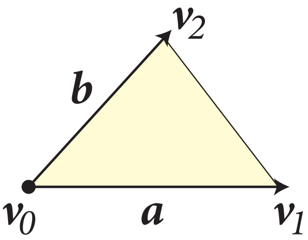


Normal vectors are *perpendicular* to the surface and have *unit* length.

Three vertex positions, $\boldsymbol{v}_0$, $\boldsymbol{v}_1$, and $\boldsymbol{v}_2$ define a triangle.



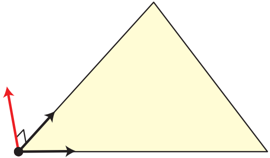
Compute the vectors along two of its edges.



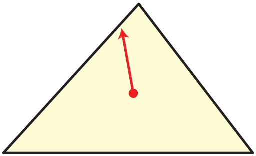
$$\boldsymbol{a} = \boldsymbol{v}_1 - \boldsymbol{v}_0$$

$$\boldsymbol{b} = \boldsymbol{v}_2 - \boldsymbol{v}_0$$

Normalize *a* and *b*, and take their cross product.

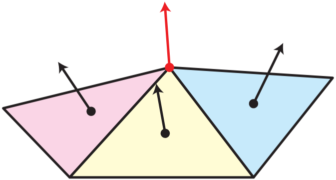


$$\boldsymbol{n} = \frac{\boldsymbol{a}}{|\boldsymbol{a}|} \times \frac{\boldsymbol{b}}{|\boldsymbol{b}|}$$



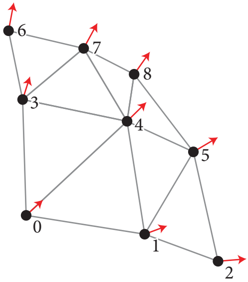
This vector ***n*** will be perpendicular to the triangle at *all* points.

Compute the normal of a *vertex* by averaging the normals of all *triangles* adjacent to that vertex.



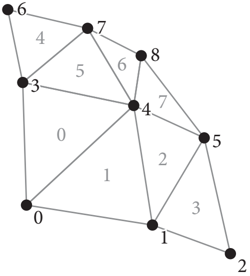
$$\boldsymbol{n}_v = \frac{\boldsymbol{n}_a + \boldsymbol{n}_c + \boldsymbol{n}_c}{|\boldsymbol{n}_a + \boldsymbol{n}_c + \boldsymbol{n}_c|}$$

3D position and normal for each vertex



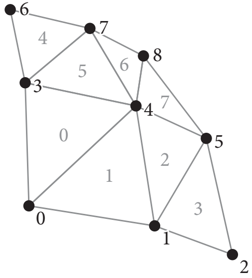
i	v_x	v_y	v_z	n_x	n_y	n_z
0	0.382	0.000	0.923	0.545	0.191	0.816
1	0.707	0.000	0.707	0.737	0.194	0.646
2	0.923	0.000	0.382	0.816	0.191	0.545
3	0.353	0.382	0.853	0.499	0.436	0.748
4	0.653	0.382	0.653	0.649	0.383	0.656
5	0.853	0.382	0.353	0.788	0.315	0.527
6	0.270	0.707	0.653	0.464	0.548	0.695
7	0.500	0.707	0.500	0.548	0.554	0.625
8	0.653	0.707	0.270	0.695	0.548	0.464

Vertex indices for each triangle



	i_0	i_1	i_2
0	3	0	4
1	4	0	1
2	4	1	5
3	5	1	2
4	6	3	7
5	7	3	4
6	7	4	8
7	8	4	5

Triangle Strips



	i_0	i_1	i_2	i_3	i_4	i_5
0	3	0	4	1	5	2
1	6	3	7	4	8	5

An alternative, efficient mechanism for defining triangles.

OpenGL Geometry

Vertex attributes are usually *interleaved* into an array of structures.

```
struct vert
{
    GLfloat v[3];
    GLfloat n[3];
};

struct vert v[...] = { ... };
```

OpenGL Geometry

So too with element indices. This example demonstrates triangles rather than triangle strips.

```
struct elem
{
    GLushort i[3];
};

struct elem e[...] = { ... };
```

Vertex Buffer Objects

Vertices are copied to the GPU via a *buffer object* bound to the *array buffer*.

```
GLuint vbo;
glGenBuffers(1, &vbo);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, sizeof (v), v);
```

Element Buffer Objects

Elements are copied to the GPU via a buffer object bound to the *element array buffer*.

```
GLuint ebo;
glGenBuffers(1, &ebo);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ebo);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof (e), e);
```

Rendering VBOs

To use a vertex buffer, OpenGL must be know the format of the data stored within it.

```
size_t s = sizeof (GLfloat);
glVertexPointer(3, GL_FLOAT, s * 6, (GLvoid *) (     0));
glNormalPointer(   GL_FLOAT, s * 6, (GLvoid *) (sz * 3));
```

Rendering VBOs

And, like everything else in OpenGL, the buffers must be *enabled* prior to their use.

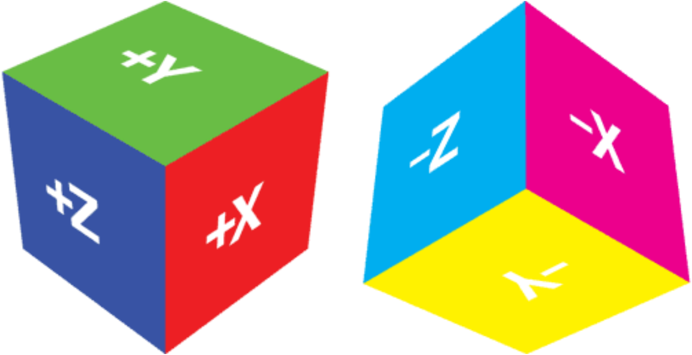
```
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_NORMAL_ARRAY);
```

Rendering VBOs

With the setup complete, we can finally render the geometry stored in our buffer objects.

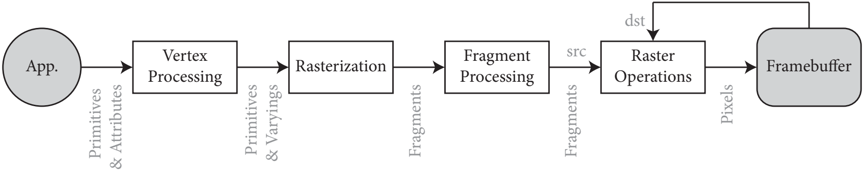
```
glDrawElements(GL_TRIANGLES, n, GL_UNSIGNED_SHORT, 0);
```

Example: 3D Reference Cube



cube.c – cube.pdf

The Basic 3D Graphics Pipeline

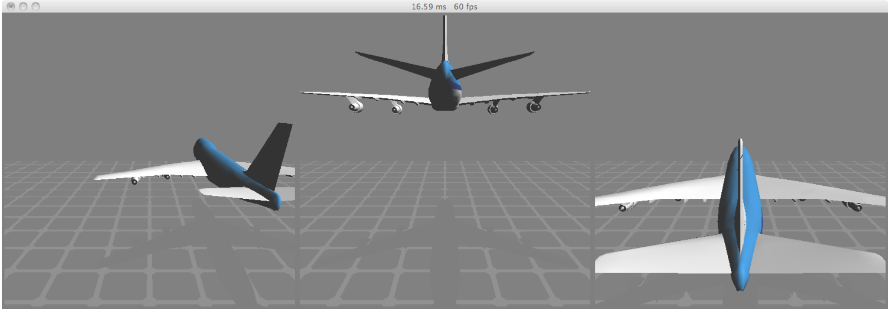


Vertex Processing

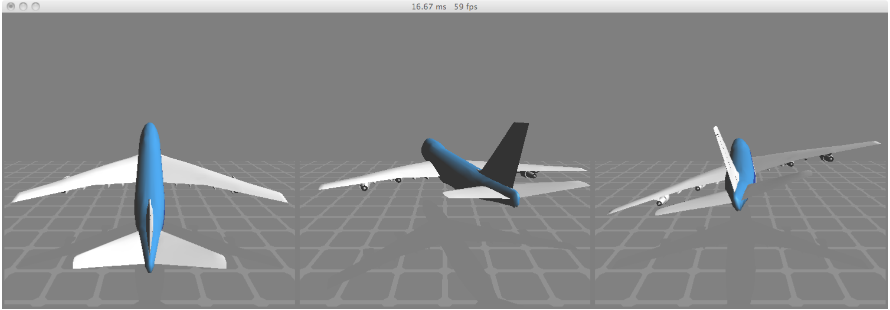
Vertex Processing prepares primitives and attributes for rasterization.

- Transformation
- Lighting

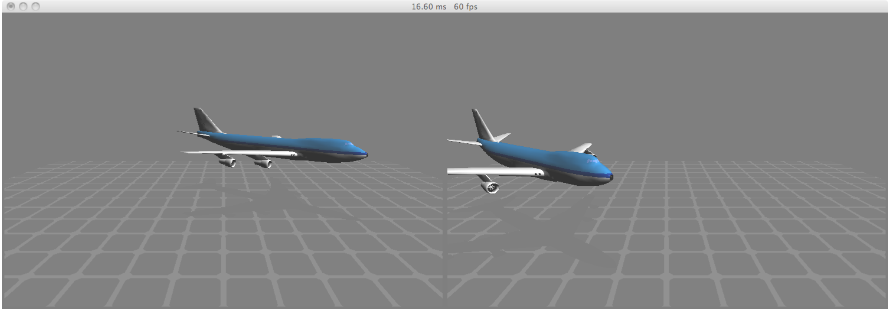
Translation Transformation



Rotation Transformation



Composition of Transformations



Relevant OpenGL

- `glLoadIdentity();`
- `glTranslatef(x, y, z);`
- `glRotatef(a, x, y, z);`
- `glScalef(x, y, z);`

We will examine these in great detail in an upcoming lecture.

Lighting

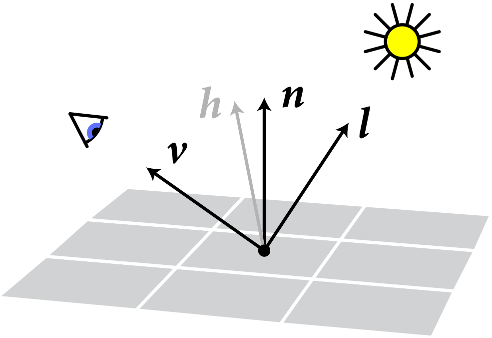
The bulk of Computer Graphics is devoted to simulating the appearance of light interacting with surface material.

Real-time 3D uses a highly simplified approximation.

Lighting

There are a small number of significant variables in real-time, fixed-function lighting:

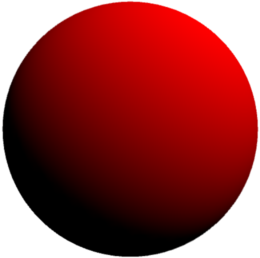
1. The unit vector $\boldsymbol{n}$ normal to the surface.
2. The unit vector $\boldsymbol{l}$ toward the light.
3. The unit vector $\boldsymbol{v}$ toward the viewer.
4. The diffuse $\boldsymbol{m}_d$ and specular $\boldsymbol{m}_s$ material properties.



The “half-angle” vector ***h*** is half-way between the light vector ***l*** and the view vector ***v***.

Diffuse Lighting

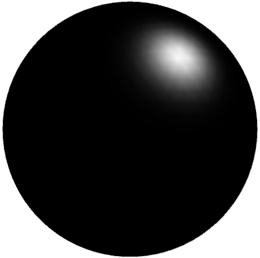
Diffuse lighting models matte surfaces.



$$\boldsymbol{c}_d = \boldsymbol{m}_d \, (\boldsymbol{n} \cdot \boldsymbol{l})$$

Specular Lighting

Specular lighting models glossy surfaces.

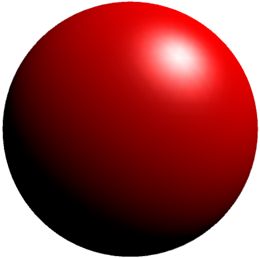


$$\boldsymbol{c}_s = \boldsymbol{m}_s \, (\boldsymbol{n} \cdot \boldsymbol{h})^{\alpha}$$

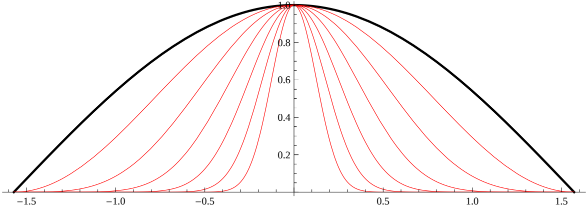
The *specular exponent* α determines the tightness of the specularity.

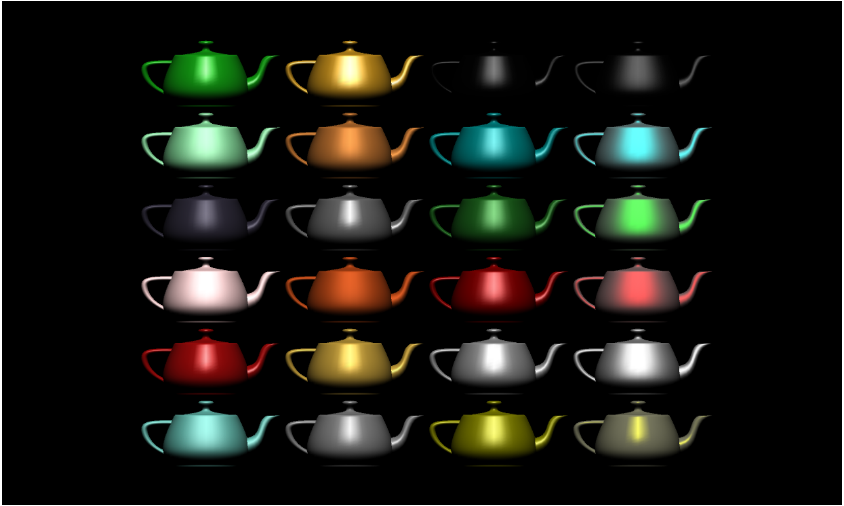
Diffuse plus Specular

Most materials exhibit both diffuse *and* specular properties, so one usually uses both at the same time.



$$\boldsymbol{c} = \boldsymbol{m}_d\,(\boldsymbol{n}\cdot\boldsymbol{l}) + \boldsymbol{m}_s\,(\boldsymbol{n}\cdot\boldsymbol{h})^{\alpha}$$





The teapot demo is a reproduction of Color Plate 17 from the OpenGL Programming Guide. The materials are purportedly:

emerald	brass	black plastic	black rubber
jade	bronze	cyan plastic	cyan rubber
obsidian	chrome	green plastic	green rubber
pearl	copper	red plastic	red rubber
ruby	gold	white plastic	white rubber
turquoise	silver	yellow plastic	yellow rubber

This demonstration reveals more about the *limitations* of the OpenGL fixed function pipeline than it does about its power.

Relevant OpenGL

- `glEnable(GL_LIGHTING);`
- `glEnable(GL_LIGHT0);`
- `glLightfv(GL_LIGHT0, GL_POSITION, p);`
- `glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, md);`
- `glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, ms);`

Coming up…

We’ll finish discussing the Basic Pipeline.