CIS 441/541: Introduction to Computer Graphics
Lecture 15: More OpenGL
Announcements

- Midterm coming up in mid-November
  - Date announced on Monday
- 541 students: emails coming about projects
Corrections, Corrections, Corrections
Correction #1
Let's do an example

- **Input parameters:** \( (\alpha, n, f) = (90, 5, 10) \)

More points:
- \((0,7,-4,1) = (0,7,8,4) = (0, 1.75, 2)\)
- \((0,7,-5,1) = (0,7,5,5) = (0, 1.4, 1)\)
- \((0,7,-6,1) = (0,7,2,6) = (0, 1.16, 0.33)\)
- \((0,7,-8,1) = (0,7,-4,8) = (0, 0.88, -0.5)\)
- \((0,7,-10,1) = (0,7,-10,10) = (0, 0.7, -1)\)
- \((0,7,-11,1) = (0,7,-13,11) = (0, .63, -1.18)\)
More points:

(0,7,-4,1) = (0,7,8,4) = (0, 1.75, 2)
(0,7,-5,1) = (0,7, 5,5) = (0, 1.4, 1)
(0,7,-6,1) = (0,7, 2,6) = (0, 1.16, 0.33)
(0,7,-8,1) = (0,7,-4,8) = (0, 0.88, -0.5)
(0,7,-10,1) = (0,7,-10,10) = (0, 0.7, -1)
(0,7,-11,1) = (0,7,-13,11) = (0, .63, -1.18)
Correction #2

- Matrices for project 1F in slides/handout
Correct answers given for
GetCamera(0, 1000)

Camera Frame: U = 0, 0.707107, -0.707107
Camera Frame: V = -0.816497, 0.408248, 0.408248
Camera Frame: W = -0.7071068, 0.5773503, 0.5773503
Camera Frame: O = 0, 0, 40
Camera Transform
(0.0000000, -0.8164966, 0.5773503, 0.0000000)
(0.7071068, 0.4082483, 0.5773503, 0.0000000)
(-0.7071068, 0.4082483, 0.5773503, 0.0000000)
(0.0000000, 0.0000000, -69.2820323, 1.0000000)
View Transform
(3.7320508, 0.0000000, 0.0000000, 0.0000000)
(0.0000000, 3.7320508, 0.0000000, 0.0000000)
(0.0000000, 0.0000000, 1.0512821, -1.0000000)
(0.0000000, 0.0000000, 10.2564103, 0.0000000)
Transformed 37.1132, 37.1132, 37.1132, 1 to 0, 0, 1
Transformed -75.4701, -75.4701, -75.4701, 1 to 0, 0, -1

(at least that’s what Hank thinks)
Correct answers given for GetCamera(0, 1000) in prompt

Camera is:
N: 5, F: 200
angle: 0.523599
position: 0, 40, 40
focus: 0, 0, 0
up: 0, 1, 0
Camera Frame: U = 1, 0, 0
Camera Frame: V = 0, 0.707107, -0.707107
Camera Frame: W = 0, 0.707107, 0.707107
Camera Frame: O = 0, 40, 40

Camera Transform
(1.0000000 0.0000000 0.0000000 0.0000000)
(0.0000000 0.7071068 0.7071068 0.0000000)
(0.0000000 -0.7071068 0.7071068 0.0000000)
(0.0000000 0.0000000 -56.5685425 1.0000000)

View Transform
(3.7320508 0.0000000 0.0000000 0.0000000)
(0.0000000 3.7320508 0.0000000 0.0000000)
(0.0000000 0.0000000 1.0512821 -1.0000000)
(0.0000000 0.0000000 10.2564103 0.0000000)

Transformed 0, 36.4645,36.4645, 1 to 500, 500,1
Transformed 0, -101.421,-101.421, 1 to 500, 500,-1

(at least that’s what Hank thinks)
this also matches the 1F prompt
Correction #3: is +1 the front of the z-buffer?

FALSE?

TRUE

IMPORTANT: 1E had Z=-1 as the front. 1F has Z=+1 as the front. This means your depth test needs to switch from “<” to “>” when considering a pixel. I apologize for this change.
Correction #4

- Device transform
Image Space to Device Space

- \((x, y, z) \rightarrow (x', y', z')\), where
  - \(x' = \frac{n(x+1)}{2}\)
  - \(y' = \frac{m(y+1)}{2}\)
  - \(z' = z\)
  - (for an \(n \times m\) image)

- **Matrix:**
  
  \[
  \begin{pmatrix}
  x' & 0 & 0 & 0 \\
  0 & y' & 0 & 0 \\
  0 & 0 & z' & 0 \\
  0 & 0 & 0 & 1 \\
  \end{pmatrix}
  \]

  **THIS IS NOT A WELL-FORMED MATRIX**
How do we transform from Image Space to Device Space?

- What should we do to Z coordinates?
  - Nothing!

- What should we do to X coordinates?
  - Answer: add 1 and multiply by width/2
  - Or: multiply by width/2 and add width/2

- What should we do to Y coordinates?
  - Answer: add 1 and multiply by height/2
  - Or: multiply by height/2 and add height/2

- What to do when width != height?
Matrix to scale X by 2

\[
\begin{bmatrix}
2 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
= \begin{bmatrix}
2x \\
y \\
z \\
1
\end{bmatrix}
\]
Matrix to scale Y by 2

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
x & y & z & 1 \\
0 & 2 & 0 & 0 \\
0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
= \begin{bmatrix}
x & 2y & z & 1 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Matrix to scale X by 3 and Y by 2:

\[
\begin{bmatrix}
3 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1 \\
\end{bmatrix} =
\begin{bmatrix}
3x \\
2y \\
z \\
1 \\
\end{bmatrix}
\]
Matrix to translate X by 1

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
x & y & z & 1 \\
\end{bmatrix}
\begin{bmatrix}
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 \\
\end{bmatrix}
= \begin{bmatrix}
x + 1 & y & z & 1 \\
\end{bmatrix}
\]
Matrix to translate Y by 2

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 2 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
= 
\begin{bmatrix}
x \\
y+2 \\
z \\
1
\end{bmatrix}
\]
Matrix to translate X by 3 and Y by 2

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
3 & 2 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
= \begin{bmatrix}
x+3 \\
y+2 \\
z \\
1
\end{bmatrix}
\]
How do we transform from Image Space to Device Space?

- What should we do to Z coordinates?
  - Nothing!

- What should we do to X coordinates?
  - Answer: add 1 and multiply by width/2
  - Or: multiply by width/2 and add width/2

- What should we do to Y coordinates?
  - Answer: add 1 and multiply by height/2
  - Or: multiply by height/2 and add height/2

- What to do when width ≠ height?
Matrix add 1 and multiply by W

\((W = \text{width}/2)\)

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
W & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
W & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
W & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
= 
\begin{bmatrix}
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 \\
W & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\]
Multiply by \( W \) and add \( W \)
\((W = \text{width}/2)\)

\[
\begin{bmatrix}
W & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
W & 0 & 0 & 1 \\
\end{bmatrix}
= \begin{bmatrix}
W & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
W & 0 & 0 & 1 \\
\end{bmatrix}
\]
Both approaches lead to the same matrix

\[
\begin{bmatrix}
W & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 \\
\end{bmatrix}
= 
\begin{bmatrix}
W & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
W & 0 & 0 & 1 \\
\end{bmatrix}
\]
Pick scale factor (width or height).

Easy if they are the same

\[
\begin{bmatrix}
W & 0 & 0 & 0 \\
0 & W & 0 & 0 \\
0 & 0 & 1 & 0 \\
W & W & 0 & 1 \\
\end{bmatrix}
\]
Can You Do Examples of Matrix Transforms?
Example

- GetCamera(0, 1000)
  - Near = 5
  - Far = 200
  - Position = (0, 40, 40)
  - Focus = (0, 0, 0)
  - Up = (0, 1, 0)
  - Angle = 30 degrees
Steps

- Build camera frame
- Calculate camera transform
- Calculate view transform
- Calculate device transform
- Compose matrices
Steps

- Build camera frame
- Calculate camera transform
- Calculate view transform
- Calculate device transform
- Compose matrices
Camera frame summarized

- \( O = \) camera position
- \( v_1 = \) Up \( \times (O\text{-focus}) \)
- \( v_2 = (O\text{-focus}) \times v_1 \)
- \( v_3 = O\text{-focus} \)

GetCamera(0, 1000)

- Near = 5
- Far = 200
- Position = (0, 40, 40)
- Focus = (0, 0, 0)
- Up = (0, 1, 0)
- Angle = 30 degrees

\[
O = (0, 40, 40) \\
v_3 = (0, 40, 40) - (0, 0, 0) = (0, 40, 40) \Rightarrow (0, 0.707, 0.707) \\
v_1 = (0, 1, 0) \times (0, 0.707, 0.707) = (0.707, 0, 0) \Rightarrow (1, 0, 0) \\
v_2 = (0, 0.707, 0.707) \times (1, 0, 0) = (0, 0.707, -0.707)
\]
Steps

- Build camera frame
- Calculate camera transform
- Calculate view transform
- Calculate device transform
- Compose matrices
Solving the Camera Transform

\[
\begin{bmatrix}
e_{1,1} & e_{1,2} & e_{1,3} & 0 \\
e_{2,1} & e_{2,2} & e_{2,3} & 0 \\
e_{3,1} & e_{3,2} & e_{3,3} & 0 \\
e_{4,1} & e_{4,2} & e_{4,3} & 1 \\
\end{bmatrix}
= 
\begin{bmatrix}
v_{1.x} & v_{2.x} & v_{3.x} & 0 \\
v_{1.y} & v_{2.y} & v_{3.y} & 0 \\
v_{1.z} & v_{2.z} & v_{3.z} & 0 \\
v_{1.t} & v_{2.t} & v_{3.t} & 1 \\
\end{bmatrix}
\]

Where \( t = (0,0,0) - O \)

How do we know?: Cramer’s Rule + simplifications

Want to derive?:

http://www.idav.ucdavis.edu/education/
GraphicsNotes/Camera-Transform/Camera-Transform.html
Solving the Camera Transform

\[
\begin{bmatrix}
  v1.x & v2.x & v3.x & 0 \\
v1.y & v2.y & v3.y & 0 \\
v1.z & v2.z & v3.z & 0 \\
v1.t & v2.t & v3.t & 1
\end{bmatrix}
\]

Where \( t = (0,0,0) - O = (0,0,0)-(0,40,40) = (0, -40, -40) \)

\[ O=(0,40,40) \]
\[ v3=(0,40,40)-(0,0,0) = (0,40,40) \rightarrow (0, 0.707, 0.707) \]
\[ v1 = (0, 1, 0) \times (0, 0.707, 0.707) = (0.707,0,0) \rightarrow (1,0,0) \]
\[ v2 = (0, 0.707, 0.707) \times (1, 0, 0) = (0, 0.707, -0.707) \]
Solving the Camera Transform
(repeat for students looking at PDF)

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0.707 & 0.707 & 0 \\
0 & -0.707 & 0.707 & 0 \\
0 & 0 & -56.57 & 1 \\
\end{bmatrix} \begin{bmatrix}
v1.x \\
v2.x \\
v3.x \\
0 \\
\end{bmatrix} = \begin{bmatrix}
v1.y \\
v2.y \\
v3.y \\
0 \\
\end{bmatrix} = \begin{bmatrix}
v1.z \\
v2.z \\
v3.z \\
0 \\
\end{bmatrix} = \begin{bmatrix}
v1 \cdot t \\
v2 \cdot t \\
v3 \cdot t \\
1 \\
\end{bmatrix}
\]

Where \( t = (0,0,0) - O = (0,0,0) - (0,40,40) = (0, -40, -40) \)

\[O=(0,40,40)\]
\[v3=(0,40,40)-(0,0,0) = (0,40,40) \Rightarrow (0, 0.707, 0.707)\]
\[v1 = (0, 1, 0) \times (0, 0.707, 0.707) = (0.707,0,0) \Rightarrow (1,0,0)\]
\[v2 = (0, 0.707, 0.707) \times (1, 0, 0) = (0, 0.707, -0.707)\]
Steps

- Build camera frame
- Calculate camera transform
- **Calculate view transform**
- Calculate device transform
- Compose matrices
The View Transformation

- Input parameters: \((\alpha, n, f)\)
- Transforms view frustum to image space cube
  - View frustum: bounded by viewing pyramid and planes \(z = -n\) and \(z = -f\)
  - Image space cube: \(-1 \leq u, v, w \leq 1\)
    
    \[
    \begin{bmatrix}
    \cot(\alpha/2) & 0 & 0 & 0 \\
    0 & \cot(\alpha/2) & 0 & 0 \\
    0 & 0 & (f+n)/(f-n) & -1 \\
    0 & 0 & 2fn/(f-n) & 0 \\
    \end{bmatrix}
    \]

- Cotangent = \(1/\text{tangent}\)
**The View Transformation**

- **Input parameters:** $(\alpha, n, f)$
- **GetCamera(0, 1000)**
  - Near = 5
  - Far = 200
  - Angle = 30 degrees

<table>
<thead>
<tr>
<th>$\cot(\alpha/2) = 3.73$</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\cot(\alpha/2) = 3.73$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>$(f+n)/(f-n) = 1.051$</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>$2fn/(f-n) = 10.26$</td>
<td>0</td>
</tr>
</tbody>
</table>
Steps

- Build camera frame
- Calculate camera transform
- Calculate view transform
- Calculate device transform
- Compose matrices
WRONG:
Image Space to Device Space

- \((x, y, z) \rightarrow (x', y', z')\), where
  - \(x' = n \times (x+1)/2\)
  - \(y' = m \times (y+1)/2\)
  - \(z' = z\)
- \((\text{for an } n \times m \text{ image})\)

- **Matrix:**
  - \((x' \ 0 \ 0 \ 0)\)
  - \((0 \ y' \ 0 \ 0)\)
  - \((0 \ 0 \ z' \ 0)\)
  - \((0 \ 0 \ 0 \ 1)\)
(x, y, z) \rightarrow (x', y', z'), \text{ where}

\[ x' = n \cdot (x+1)/2 = nx/2 + n/2 \]
\[ y' = n \cdot (y+1)/2 = ny/2 + n/2 \]
\[ z' = z = z \]

(for an n x n image)

- n, m = 1000 for 1F

Matrix:

\[
\begin{pmatrix}
\frac{n}{2} & 0 & 0 & 0 \\
0 & \frac{n}{2} & 0 & 0 \\
0 & 0 & 1 & 0 \\
\frac{n}{2} & \frac{n}{2} & 0 & 1
\end{pmatrix}
= \begin{pmatrix}
500 & 0 & 0 & 0 \\
0 & 500 & 0 & 0 \\
0 & 0 & 1 & 0 \\
500 & 500 & 0 & 1
\end{pmatrix}
\]
Steps

- Build camera frame
- Calculate camera transform
- Calculate view transform
- Calculate device transform
- Compose matrices
Done by using Matrix class

Matrix for our example:

\[
\begin{pmatrix}
1866.03 & 0 & 0 & 0 \\
-353.55 & 965.92 & 0.743 & -0.707 \\
-353.55 & -1673.03 & 0.743 & -0.707 \\
28284.3 & 28284.3 & -49.2 & 56.57
\end{pmatrix}
\]
Floating point precision

#include <iostream>
#include <float.h>

using std::cerr;
using std::endl;

#include <limits>
#include <math.h>

int main()
{
    float X = FLT_MAX;
    float Y = -X;
    float Z = -1;
    float W1 = (X+Y)+Z;
    float W2 = X+(Y+Z);
    fprintf(stderr, "W1 = %f, W2 = %f\n", W1, W2);
}

C02LN00GFD58:Downloads hank$ g++ tmp.C
C02LN00GFD58:Downloads hank$ ./a.out
W1 = -1.000000, W2 = 0.000000
New policy for pixel differences

Hi Everyone,

It is becoming clear that floating point differences are leading to small differences in output.

So, new policy:
1D: 0 pixels different for full credit (no change)
1E: <= 20 pixels different for full credit. These 20 pixels can be from either of the two posted baselines.
1F: <= 100 pixels different for full credit. (There is an increase because the transformations will involve more floating point math, and thus more differences.)

I apologize for the confusion, and greatly appreciate the folks who have been providing evidence for 1E that we cannot continue to all get the same answer.

Best,
Hank
Goal: add arbitrary camera positions

Extend your project1E code

Re-use:
  proj1e_geometry.vtk available on web (9MB), “reader1e.cxx”, “shading.cxx”.

No Cmake, project1F.cxx

New: Matrix.cxx, Camera.cxx
Project #1F, expanded

- Matrix.cxx: complete

- Methods:

```cpp
class Matrix
{
    public:
        double A[4][4];

        void TransformPoint(const double *ptIn, double *ptOut);
        static Matrix ComposeMatrices(const Matrix &, const Matrix &);
        void Print(ostream &o);
};
```
Camera.cxx: you work on this

class Camera
{
    public:
    double          near, far;
    double          angle;
    double          position[3];
    double          focus[3];
    double          up[3];

    Matrix  ViewTransform(void) {;};
    Matrix  CameraTransform(void) {;};
    Matrix  DeviceTransform(void) {;};
    // Will probably need something for calculating Camera Frame as well
};

Also: GetCamera(int frame, int nFrames)
Project #1F, deliverables

- Same as usual, but times 4
  - 4 images, corresponding to
    - GetCamera(0, 1000)
    - GetCamera(250, 1000)
    - GetCamera(500, 1000)
    - GetCamera(750, 1000)

- If you want:
  - Generate all thousand images, make a movie
    - Can discuss how to make a movie if there is time
vector<Triangle> t = GetTriangles();
AllocateScreen();
for (int i = 0 ; i < 1000 ; i++)
{
    InitializeScreen();
    Camera c = GetCamera(i, 1000);
    CalculateShading();
    TransformTrianglesToDeviceSpace(); // involves setting up and applying matrices
    //… if you modify vector<Triangle> t,
    // remember to undo it later

    RenderTriangles()
    SaveImage();
}
Project #1F, pitfalls

- All vertex multiplications use 4D points. Make sure you send in 4D points for input and output, or you will get weird memory errors.
  - Make sure you divide by w.

- Your Phong lighting assumed a view of (0,0,-1). The view will now be changing with each render and you will need to incorporate that view direction in your rendering.
People often get a matrix confused with its transpose. Use the method Matrix::Print() to make sure the matrix you are setting up is what you think it should be. Also, remember the points are left multiplied, not right multiplied.

Regarding multiple renderings:
- Don’t forget to initialize the screen between each render
- If you modify the triangle in place to render, don’t forget to switch it back at the end of the render
Goal: add arbitrary camera positions
Project 1F: Making a Movie

- You can also generate 1000 images and use a movie encoder to make a movie (i.e., ffmpeg to make a mpeg)
  - Could someone post a how to?
Project 1F: Shading

- We used viewDirection = (0, 0, -1) for 1E
- For 1F, we will do it right:
  - Prior to transforming, you can calculate viewDirection
    - triangle vertex minus camera position
    - then call CalculateShading with correct viewDir
    - then associate shading value as a scale on the triangle
    - and then LERP that shading value across scanline
Practical Approach

- Process objects one at a time in the order they are generated by the application
  - Can consider only local lighting
- Pipeline architecture
  - All steps can be implemented in hardware on the graphics card
Vertex Processing

- Much of the work in the pipeline is in converting object representations from one coordinate system to another
  - Object coordinates
  - Camera (eye) coordinates
  - Screen coordinates
- Every change of coordinates is equivalent to a matrix transformation
- Vertex processor also computes vertex colors
Primitive Assembly

Vertices must be collected into geometric objects before clipping and rasterization can take place
- Line segments
- Polygons
- Curves and surfaces
Clipping

Just as a real camera cannot “see” the whole world, the virtual camera can only see part of the world or object space. Objects that are not within this volume are said to be clipped out of the scene.
Rasterization

- If an object is not clipped out, the appropriate pixels in the frame buffer must be assigned colors
- Rasterizer produces a set of fragments for each object
- Fragments are "potential pixels"
  - Have a location in frame buffer
  - Color and depth attributes
- Vertex attributes are interpolated over objects by the rasterizer
Fragment Processing

- Fragments are processed to determine the color of the corresponding pixel in the frame buffer.
- Colors can be determined by texture mapping or interpolation of vertex colors.
- Fragments may be blocked by other fragments closer to the camera - Hidden-surface removal.
The Programmer’s Interface

- Programmer sees the graphics system through a software interface: the Application Programmer Interface (API)
API Contents

• Functions that specify what we need to form an image
  - Objects
  - Viewer
  - Light Source(s)
  - Materials

• Other information
  - Input from devices such as mouse and keyboard
  - Capabilities of system
Object Specification

• Most APIs support a limited set of primitives including
  - Points (0D object)
  - Line segments (1D objects)
  - Polygons (2D objects)
  - Some curves and surfaces
    • Quadrics
    • Parametric polynomials

• All are defined through locations in space or vertices
Example

```c
glBegin(GL_POLYGON);
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(0.0, 1.0, 0.0);
    glVertex3f(0.0, 0.0, 1.0);
glEnd();
```

type of object

location of vertex

dead of object definition
Lights and Materials

• Types of lights
  - Point sources vs distributed sources
  - Spot lights
  - Near and far sources
  - Color properties

• Material properties
  - Absorption: color properties
  - Scattering
    • Diffuse
    • Specular
NEW MATERIAL
OpenGL Libraries

• OpenGL core library
  - OpenGL32 on Windows
  - GL on most unix/linux systems (libGL.a)

• OpenGL Utility Library (GLU)
  - Provides functionality in OpenGL core but avoids having to rewrite code

• Links with window system
  - GLX for X window systems
  - WGL for Windows
  - AGL for Macintosh

The University of New Mexico
• OpenGL Utility Toolkit (GLUT)
  - Provides functionality common to all window systems
    • Open a window
    • Get input from mouse and keyboard
    • Menus
    • Event-driven
  - Code is portable but GLUT lacks the functionality of a good toolkit for a specific platform
    • No slide bars

• <GLUT no longer well maintained, we will use VTK>
OpenGL Architecture

Immediate Mode

Polynomial Evaluator

Display List

Per Vertex Operations & Primitive Assembly

Rasterization

Texture Memory

Pixel Operations

Per Fragment Operations

Frame Buffer

geometry pipeline
OpenGL Functions

- Primitives
  - Points
  - Line Segments
  - Polygons
- Attributes
- Transformations
  - Viewing
  - Modeling
- Control (GLUT)
- Input (GLUT)
- Query

} VTK
OpenGL State

• OpenGL is a state machine

• OpenGL functions are of two types
  - Primitive generating
    • Can cause output if primitive is visible
    • How vertices are processed and appearance of primitive are controlled by the state
  - State changing
    • Transformation functions
    • Attribute functions
Lack of Object Orientation

• OpenGL is not object oriented so that there are multiple functions for a given logical function
  - `glVertex3f`
  - `glVertex2i`
  - `glVertex3dv`

• Underlying storage mode is the same

• Easy to create overloaded functions in C++ but issue is efficiency
OpenGL function format

glVertex3f(x,y,z)

- function name
- x,y,z are floats
- belongs to GL library

glVertex3fv(p)

- p is a pointer to an array
OpenGL #defines

• Most constants are defined in the include files `gl.h`, `glu.h` and `glut.h`
  - Note `#include <GL/glut.h>` should automatically include the others
  - Examples
    - `glBegin(GL_POLYGON)`
    - `glClear(GL_COLOR_BUFFER_BIT)`
• include files also define OpenGL data types: `GLfloat`, `GLdouble`, ....
A Simple Program

Generate a square on a solid background
```c
#include <GL/glut.h>
void mydisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
    glVertex2f(-0.5, -0.5);
    glVertex2f(-0.5, 0.5);
    glVertex2f(0.5, 0.5);
    glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}
int main(int argc, char** argv){
    glutCreateWindow("simple");
    glutDisplayFunc(mydisplay);
    glutMainLoop();
}
```
Event Loop

• Note that the program defines a display callback function named **mydisplay**

  - Every glut program must have a display callback

  - The display callback is executed whenever OpenGL decides the display must be refreshed, for example when the window is opened

  - The **main** function ends with the program entering an event loop

  > VTK will be similar ... callback issued to render geometry
Defaults

- **simple.c** is too simple
- Makes heavy use of state variable default values for
  - Viewing
  - Colors
  - Window parameters
- Next version will make the defaults more explicit
How to make a graphics program?

- Need to create a window
  - This window contains a “context” for OpenGL to render in.
- Need to be able to deal with events/interactions
- Need to render graphics primitives
  - OpenGL!
Windows and Events

• Creating windows and dealing with events varies from platform to platform.
• “Hello World” with X-Windows.

Compile with:

```
gcc -L/usr/X11R6/lib -lX11 hello-x.c -o hello-x
```

```c
#include <X11/Xlib.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {
    Display *d;
    Window w;
    XEvent e;
    char *msg = "Hello, World!";
    int s;

    d = XOpenDisplay(NULL);
    if (d == NULL) {
        fprintf(stderr, "Cannot open display\n");
        exit(1);
    }

    s = DefaultScreen(d);
    w = XCreateSimpleWindow(d, RootWindow(d, s), 10, 10, 100, 100, 1,
                             BlackPixel(d, s), WhitePixel(d, s));
    XSelectInput(d, w, ExposureMask | KeyPressMask);
    XMapWindow(d, w);

    while (1) {
        XNextEvent(d, &e);
        if (e.type == Expose) {
            XFillRectangle(d, w, DefaultGC(d, s), 20, 20, 10, 10);
            XDrawString(d, w, DefaultGC(d, s), 10, 50, msg, strlen(msg));
        }
        if (e.type == KeyPress)
            break;
    }

    XCloseDisplay(d);
    return 0;
}
```
Windows and Events

• Creating windows and dealing with events varies from platform to platform.
• Some packages provide implementations for key platforms (Windows, Unix, Mac) and abstractions for dealing with windows and events.
• GLUT: library for cross-platform windowing & events.
  - My experiments: doesn’t work as well as it used to.
• VTK: library for visualization
Visualization with VTK

Content from: Erik Vidholm, Univ of Uppsula, Sweden
David Gobbi, Robarts Research Institute, London, Ontario, Canada
• Open source, freely available software for 3D computer graphics, image processing, and visualization
• Managed by Kitware Inc.
• Use C++, Tcl/Tk, Python, Java
The visualization pipeline

DATA

Visualization algorithms

FILTER

MAPPING

Interactive feedback

DISPLAY
We will replace these and write our own GL calls.

from vtkpython import *
cone = vtkConeSource()
cone.SetResolution(10)

coneMapper = vtkPolyDataMapper()
coneMapper.SetInput(cone.GetOutput())

coneActor = vtkActor()
coneActor.SetMapper(coneMapper)

ren = vtkRenderer()
ren.AddActor(coneActor)

renWin = vtkRenderWindow()
renWin.SetWindowName("Cone")
renWin.SetSize(300,300)
renWin.AddRenderer(ren)
iren = vtkRenderWindowInteractor()
iren.SetRenderWindow(renWin)
iren.Initialize()
iren.Start()
How to make a graphics program?

• Need to create a window
  - This window contains a “context” for OpenGL to render in.

• Need to be able to deal with events/interactions

• Need to render graphics primitives
  - OpenGL!

Borrow Build
OpenGL Functions

- Primitives
  - Points
  - Line Segments
  - Polygons
- Attributes
- Transformations
  - Viewing
  - Modeling
- Control (VTK)
- Input (VTK)
- Query

Today

next week
First OpenGL programs

- Remember: none of these programs have windowing or events
- They contain just the code to put primitives on the screen, with lighting and colors.
class vtk441PolyDataMapper : public vtkOpenGLPolyDataMapper
{
public:
    static vtk441PolyDataMapper *New();
    virtual void RenderPiece(vtkRenderer *ren, vtkActor *act)
{
    float ambient[3] = { 1, 1, 1 };
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
    glBegin(GL_TRIANGLES);
    glVertex3f(0,0,0);
    glVertex3f(0,1,0);
    glVertex3f(1,1,0);
    glEnd();
};
class vtk441PolyDataMapper : public vtkOpenGLPolyDataMapper
{
  public:
    static vtk441PolyDataMapper *New();
    virtual void RenderPiece(vtkRenderer *ren, vtkActor *act)
    {
      glEnable(GL_COLOR_MATERIAL);
      float ambient[3] = { 1, 1, 1 };
      glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
      glBegin(GL_TRIANGLES);
      glColor3ub(0, 0, 255);
      glVertex3f(0,0,0);
      glVertex3f(0,1,0);
      glVertex3f(1,1,0);
      glEnd();
    }
};
Both glEnable and glDisable take a single argument, *cap*, which can assume one of the following values:

- **GL_BLEND**
  If enabled, blend the computed fragment color values with the values in the color buffers. See glBlendFunc.

- **GL_CULL_FACE**
  If enabled, cull polygons based on their winding in window coordinates. See glCullFace.

- **GL_DEPTH_TEST**
  If enabled, do depth comparisons and update the depth buffer. Note that even if the depth buffer exists and the depth mask is non-zero, the depth buffer is not updated if the depth test is disabled. See glDepthFunc and glDepthRange.

- **GL_DITHER**
  If enabled, dither color components or indices before they are written to the color buffer.

- **GL_POLYGON_OFFSET_FILL**
  If enabled, an offset is added to depth values of a polygon's fragments produced by rasterization. See glPolygonOffset.

- **GL_SAMPLE_ALPHA_TO_COVERAGE**
  If enabled, compute a temporary coverage value where each bit is determined by the alpha value at the corresponding sample location. The temporary coverage value is then ANDed with the fragment coverage value.

- **GL_SAMPLE_COVERAGE**
  If enabled, the fragment's coverage is ANDed with the temporary coverage value. If GL_SAMPLE_COVERAGE_INVERT is set to GL_TRUE, invert the coverage value. See glSampleCoverage.

- **GL_SCISSOR_TEST**
  If enabled, discard fragments that are outside the scissor rectangle. See glScissor.

- **GL_STENCIL_TEST**
  If enabled, do stencil testing and update the stencil buffer. See glStencilFunc and glStencilOp.
First OpenGL programs

class vtk441PolyDataMapper : public vtkOpenGLPolyDataMapper
{
  public:
    static vtk441PolyDataMapper *New();
    virtual void RenderPiece(vtkRenderer *ren, vtkActor *act)
    {
      glEnable(GL_COLOR_MATERIAL);
      float ambient[3] = { 1, 1, 1 };
      glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
      glBegin(GL_TRIANGLES);
      glColor3ub(0, 0, 255);
      glVertex3f(0, 0, 0);
      glColor3ub(0, 255, 0);
      glVertex3f(0, 1, 0);
      glColor3ub(255, 0, 0);
      glVertex3f(1, 1, 0);
      glEnd();
    }
};
Visualization use case

Why is there purple in this picture?
class vtk441PolyDataMapper : public vtkOpenGLPolyDataMapper
{
    public:
    static vtk441PolyDataMapper *New();
    virtual void RenderPiece(vtkRenderer *ren, vtkActor *act)
    {
        glEnable(GL_COLOR_MATERIAL);
        float ambient[3] = { 1, 1, 1 };
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
        glBegin(GL_TRIANGLES);
        glColor3ub(0, 0, 255);
        glVertex3f(0, 0, 0);
        glColor3ub(0, 255, 0);
        glVertex3f(0, 0, 0);
        glColor3ub(255, 0, 0);
        glVertex3f(0, 0, 0);
        glEnd();
    }
};