Outline

- Corrections, corrections, corrections
- Project 1F
- Review
- More OpenGL & Textures
- Project 2A
Corrections, corrections, corrections

Project 1F

Review

More OpenGL & Textures

Project 2A
Correction #1

- View transformation example in the lecture slides
Let's do an example

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

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 3 & -1 & 0 \\
0 & 0 & 20 & 0 & 0
\end{bmatrix}
\]

More points:

- \((0,7,-4,1) = (0,7,8,4) = (0,1.75,2)\)
- \((0,7,-5,1) = (0,7,5,5) = (0,1.4,1)\)
- \((0,7,-6,1) = (0,7,2,6) = (0,1.16,0.33)\)
- \((0,7,-8,1) = (0,7,-4,8) = (0,0.88,-0.5)\)
- \((0,7,-10,1) = (0,7,-10,10) = (0,0.7,-1)\)
- \((0,7,-11,1) = (0,7,-13,11) = (0,0.63,-1.18)\)
View Transformation

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, 0.63, −1.18)
Correction #2

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

Correct camera frames and transforms:

- Camera Frame: U = 0, 0.707107, -0.707107
- Camera Frame: V = -0.816497, 0.408248, 0.408248
- Camera Frame: W = 0.57735, 0.57735, 0.57735
- Camera Frame: O = 0, 40, 40

Correct camera transforms:

- 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)

Correct view transforms:

- 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 coordinates:

- 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
V = 0, 0.707107, -0.707107
W = 0, 0.707107, 0.707107
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
  - \(x' = n(x+1)/2\)
  - \(y' = m(y+1)/2\)
  - \(z' = z\)
  - (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

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

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
x & y & z & 1
\end{bmatrix} \begin{bmatrix}
0 & 2 & 0 & 0 \\
0 & 0 & 1 & 0
\end{bmatrix} = \begin{bmatrix}
x & 2y & z & 1
\end{bmatrix}
\]
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

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
x & y & z & 1 & 1
\end{bmatrix}
\begin{bmatrix}
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0
\end{bmatrix}
= \begin{bmatrix}
0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1
\end{bmatrix}
= \begin{bmatrix}
x+1 & y & z & 1 & 1
\end{bmatrix}
\]
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

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
3 & 2 & 0 & 1 & 1
\end{bmatrix}
\]

\[
\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} x+3 \\ y+2 \\ z \\ 1 \end{bmatrix}
\]
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 = width/2)

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
W & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 \\
\end{bmatrix}
= 
\begin{bmatrix}
W & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
W & 0 & 0 & 1 \\
\end{bmatrix}
\]
Multiply by \( W \) and add \( W \)

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

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

\[
\begin{bmatrix}
W & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1 \\
\end{bmatrix}
\begin{bmatrix}
W & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
= \begin{bmatrix}
W & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
W & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1 \\
\end{bmatrix}
\begin{bmatrix}
W & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 \\
\end{bmatrix}
\]
Image space to device space matrix

- Pick scale factor (width or height).
  - Easy if they are the same

\[
\begin{bmatrix}
W & 0 & 0 & 0 & 0 \\
0 & W & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
W & W & 0 & 1 & 0 \\
W & W & 0 & 1 & 0
\end{bmatrix}
\]
Outline

- Corrections, corrections, corrections
- Project 1F
- Review
- More OpenGL & Textures
- Project 2A
How do we transform?

- For a camera $C$,
  - Calculate Camera Frame
  - From Camera Frame, calculate Camera Transform
  - Calculate View Transform
  - Calculate Device Transform
  - Compose 3 Matrices into 1 Matrix ($M$)

- For each triangle $T$, apply $M$ to each vertex of $T$, then apply rasterization/zbuffer/Phong shading

```cpp
class Camera {
    public:
        double near, far;  // angle;
        double position[3];  // focus[3];
        double up[3];
};
```
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);
    TransformTrianglesToDeviceSpace(); // involves setting up and applying matrices
    //... if you modify vector<Triangle> t,
    // remember to undo it later

    RenderTriangles()
    SaveImage();
}
Correct answers given for GetCamera(0, 1000)

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
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
Outline

- Corrections, corrections, corrections
- Project 1F
- Review
- More OpenGL & Textures
- Project 2A
OpenGL Architecture
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**: `glVertex3f(x, y, z)`
- **belongs to GL library**: `glVertex3f` belongs to the GL library.
- **dimensions**: The function takes three float parameters, `x`, `y`, and `z`.
- **x, y, z are floats**: The parameters `x`, `y`, and `z` are floats.

```
glVertex3fv(p)
```

- **p is a pointer to an array**: The function `glVertex3fv` takes a pointer `p` to an array.
A Simple Program

Generate a square on a solid background
#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
Outline

- Corrections, corrections, corrections
- Project 1F
- Review
- More OpenGL & Textures
- Project 2A
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!
Creating windows and dealing with events varies from platform to platform.
“Hello World” with X-Windows.

```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;
}
```
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
- But also contains cross-platform windowing & events.
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  FILTER  MAPPING  DISPLAY

Visualization algorithms

Interactive feedback
We will replace these and write our own GL calls.

We will re-use these.

Cone.py Pipeline Diagram (type "python Cone.py" to run)

- **Source**
  - Either reads the data from a file or creates the data from scratch.

- **Mapper**
  - Moves the data from VTK into OpenGL.

- **Actor**
  - For setting colors, surface properties, and the position of the object.

- **Renderer**
  - The rectangle of the computer screen that VTK draws into.

- **Window**
  - The window, including title bar and decorations.

- **Interactor**
  - Allows the mouse to be used to interact with the data.

```python
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!
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 };  // Ambient light
      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(0,0,0);  
        glVertex3f(1,1,0);  
        glEnd();
    }
};
glEnable/glDisable: important functions

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`. 

**Name**

glEnable — enable or disable server-side GL capability

glDisable — disable server-side GL capability

**C Specification**

```c
void glEnable(GLenum cap);

Parameters

cap

Specifies a symbolic constant indicating a GL capability.

void glDisable(GLenum cap);

Parameters

cap

Specifies a symbolic constant indicating a GL capability.
```
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(0,1,0);
        glColor3ub(255, 0, 0);
        glVertex3f(1,1,0);
        glEnd();
    }
};
Textures: a better way to specify a color map

There is no purple when we use textures
Textures

- “Textures” are a mechanism for adding “texture” to surfaces.
  - Think of texture of a cloth being applied to a surface
  - Typically used in 2D form

- We will start with a 1D form, and work our way up to 2D later.
1D textures: basic idea

- Store color map on GPU as a texture
  - An array of colors

- Old color interpolation of fragment on a scanline:
  - For (int j = 0 ; j < 3 ; j++)
    - RGB[j] = leftRGB[j] + proportion*(rightRGB[j]-leftRGB[j])

- New color interpolation of fragment on a scanline:
  - textureVal = leftTextureVal
    + proportion*(rightTextureVal-leftTextureVal)
  - RGB ← textureLookup[textureVal]
Example

- Triangle with vertices with scalar values 2.9, 3.3, and 3.1.
- $T$ for 2.9 = $(2.9-3.0)/(3.25-3) = -0.4$
- $T$ for 3.1 = $(3.1-3.0)/(3.25-3) = 0.4$
- $T$ for 3.3 = $(3.3-3.0)/(3.25-3) = 1.2$
- Fragment colors come from interpolating texture coordinates and applying texture
First OpenGL Texture Program

```cpp
class vtk441PolyDataMapper : public vtkOpenGLPolyDataMapper {
public:
    static vtk441PolyDataMapper *New();

    virtual void RenderPiece(vtkRenderer *ren, vtkActor *act) {
        GLubyte Texture3[9] = {
            0, 0, 255, // blue
            255, 255, 255, // white
            255, 0, 0, // red
        };
        glTexImage1D(GL_TEXTURE_1D, 0, GL_RGB, 3, 0, GL_RGB,
                     GL_UNSIGNED_BYTE, Texture3);
        glEnable(GL_COLOR_MATERIAL);
        glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
        glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glEnable(GL_TEXTURE_1D);
        float ambient[3] = { 1, 1, 1 };
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
        glBegin(GL_TRIANGLES);
        glVertex3f(0);
        glVertex3f(0, 0, 0);
        glVertex3f(0.0);
        glVertex3f(0, 1, 0);
        glVertex3f(0, 1, 0);
        glVertex3f(1.0);
        glVertex3f(1, 1, 0);
        glEnd();
    }
};
```
Outline

- Corrections, corrections, corrections
- Project 1F
- Review
- More OpenGL & Textures
- Project 2A
We will replace these and write our own GL calls.

Cone.py Pipeline Diagram (type "python Cone.py" to run)

- **Source**
  - Data
  - Either reads the data from a file or creates the data from scratch.

- **Mapper**
  - Moves the data from VTK into OpenGL.
  - For setting colors, surface properties, and the position of the object.
  - The rectangle of the computer screen that VTK draws into.

- **Actor**
  - The window, including title bar and decorations.
  - Allows the mouse to be used to interact with the data.

- **Renderer**
  - ren = vtkRenderWindow()
  - renWin.SetWindowName("Cone")
  - renWin.SetSize(300,300)
  - renWin.AddRenderer(ren)
  - iren = vtkRenderWindowInteractor()
  - iren.SetRenderWindow(renWin)
  - iren.Initialize()
  - iren.Start()

- **Window**

- **Interactor**

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()
Project #2A (8%), Due Nov. 12th

- Goal: OpenGL program that does regular colors and textures
- New VTK-based project2A.cxx
- New CMakeLists.txt (but same as old ones)
Hints

- I recommend you “walk before you run” & “take small bites”. OpenGL can be very punishing. Get a picture up and then improve on it. Make sure you know how to retreat to your previously working version at every step.

- OpenGL “state thrashing” is common and tricky to debug.
  - Get one window working perfectly.
  - Then make the second one work perfectly.
  - Then try to get them to work together.
    - Things often go wrong, when one program leaves the OpenGL state in a way that doesn’t suit another renderer.