CIS 441/541: Introduction to Computer Graphics
Lecture 13: CW/CCW, constructing geometry
Back from Paris...

**LDAV 2014 Best Paper**

**Improved Post Hoc Flow Analysis Via Lagrangian Representations**

Alexy Agranovsky, David Camp, Christoph Garth, E. Wes Bethel, Kenneth I. Joy, Hank Childs

University of California Davis, Lawrence Berkeley National Laboratories, University of Oregon, University of Kaiserslautern
Announcements

- Midterm Nov 19\textsuperscript{th}
  - Midterms graded and returned Nov 26\textsuperscript{th}
  - Available Nov 24\textsuperscript{th}, as well as grade consultations
- OH canceled Nov 19\textsuperscript{th}
- Everything back to normal on Nov 21\textsuperscript{st}
Outline

- Review
- Project 2B
- CW/CCW
- Geometry Creation
- Midterm Review
Outline

- Review
- Project 2B
- CW/CCW
- Geometry Creation
- Midterm Review
CPU and GPU

- Most common configuration has CPU and GPU on separate dies
  - I.e., plug GPU in CPU

CPU
(typically 4-12 cores, ~10GFLOPs)

GPU
(typically 100-1000 cores, ~100GFLOPs-~1000GFLOPs)

PCIe

Peripheral Component Interconnect Express

What are the performance ramifications of this architecture?
Display lists

- Idea:
  - Send geometry and settings to GPU once, give it an identifier
  - GPU stores geometry and settings
  - Just pass the identifier for every subsequent render
Display lists in action

for (int frame = 0 ; frame < nFrames ; frame++)
{
    SetCamera(frame, nFrames);
    glBegin(GL_TRIANGLES);
    for (int i = 0 ; i < triangles.size() ; i++)
    {
        for (int j = 0 ; j < 3 ; j++)
        {
            glColor3ubv(triangles[i].colors[j]);
            glColor3fv(triangles[i].vertices[j]);
        }
        glEnd();
    }
    glEnd();
}

GLUInt displayList = glGenLists(1);
glNewList(displayList, GL_COMPILE);
glBegin(GL_TRIANGLES);
for (int i = 0 ; i < triangles.size() ; i++)
{
    for (int j = 0 ; j < 3 ; j++)
    {
        glColor3ubv(triangles[i].colors[j]);
        glColor3fv(triangles[i].vertices[j]);
    }
    glEnd();
    glEndList();
}

for (int frame = 0 ; frame < nFrames ; frame++)
{
    SetCamera(frame, nFrames);
    glCallList(displayList);
}
Outline

- Review
- Display lists
- Transforms in GL
- Project 2B
ModelView and Projection Matrices

- ModelView idea: two purposes ... model and view
  - Model: extra matrix, just for rotating, scaling, and translating geometry.
    - How could this be useful?
  - View: Cartesian to Camera transform

(We will focus on the model part of the modelview matrix now & come back to others later)
Common commands for modifying model part of ModelView matrix

- glTranslate
- glRotate
- glScale
How do transformations combine?

```c
glScale(2, 2, 2)
glTranslate(1, 0, 0)
glRotate(45, 0, 1, 0)
```

→ Rotate by 45 degrees around (0,1,0), then translate in X by 1, then scale by 2 in all dimensions.

→ (the last transformation is applied first)
dl = GenerateTireGeometry();
glCallList(dl);  // place tire at (0, 0, 0)
glTranslatef(10, 0, 0);
glCallList(dl);  // place tire at (10, 0, 0)
glTranslatef(0, 0, 10);
glCallList(dl);  // place tire at (10, 0, 10)
glTranslatef(-10, 0, 0);
glCallList(dl);  // place tire at (0, 0, 10)

each `glTranslatef` call updates the state of the ModelView matrix.
glPushMatrix, glPopMatrix

NAME

`glPushMatrix, glPopMatrix` - push and pop the current matrix stack

C SPECIFICATION

```c
void glPushMatrix( void )
```

C SPECIFICATION

```c
void glPopMatrix( void )
```

DESCRIPTION

There is a stack of matrices for each of the matrix modes. In `GL_MODELVIEW` mode, the stack depth is at least 32. In the other two modes, `GL_PROJECTION` and `GL_TEXTURE`, the depth is at least 2. The current matrix in any mode is the matrix on the top of the stack for that mode.

`glPushMatrix` pushes the current matrix stack down by one, duplicating the current matrix. That is, after a `glPushMatrix` call, the matrix on top of the stack is identical to the one below it.

`glPopMatrix` pops the current matrix stack, replacing the current matrix with the one below it on the stack.

Initially, each of the stacks contains one matrix, an identity matrix.
dl = GenerateTireGeometry();
glCallList(dl); // place tire at (0, 0, 0)
glPushMatrix();
glTranslatef(10, 0, 0);
glCallList(dl); // place tire at (10, 0, 0)
glPopMatrix();
glPushMatrix();
glTranslatef(0, 0, 10);
glCallList(dl); // place tire at (10, 0, 10) (0, 0, 10)
glPopMatrix();

Why is this useful?
Matrices in OpenGL

- OpenGL maintains matrices for you and provides functions for setting matrices.

- There are four different modes you can use:
  - Modelview
  - Projection
  - Texture
  - Color (rarely used, often not supported)

- You control the mode using `glMatrixMode`.
Matrices in OpenGL (cont’d)

- The matrices are the identity matrix by default and you can modify them by:
  - 1) setting the matrix explicitly
  - 2) using OpenGL commands for appending to the matrix
- You can have \( \geq 32 \) matrices for modelview, \( \geq 2 \) for others
**glMatrixMode**

**NAME**

`glMatrixMode` - specify which matrix is the current matrix

**C SPECIFICATION**

```c
void glMatrixMode( GLenum mode )
```

**PARAMETERS**

- `mode` Specifies which matrix stack is the target for subsequent matrix operations. Three values are accepted: `GL_MODELVIEW`, `GL_PROJECTION`, and `GL_TEXTURE`. The initial value is `GL_MODELVIEW`.

Additionally, if the `GL_ARB_imaging` extension is supported, `GL_COLOR` is also accepted.

**DESCRIPTION**

`glMatrixMode` sets the current matrix mode. `mode` can assume one of four values:

- `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.

To find out which matrix stack is currently the target of all matrix operations, call `glGet` with argument `GL_MATRIX_MODE`. The initial value is `GL_MODELVIEW`. 
How do you put the projection transformation in GL_PROJECTION?

- Two options:
  - `glFrustum()` (perspective projection)
  - `glOrtho()` (orthographic projection)
Outline

- Review
- Project 2B
- CW/CCW
- Geometry Creation
- Midterm Review
Goal: modify ModelView matrix to create dog out of spheres and cylinders

New code skeleton: “project2B.cxx”

No geometry file needed.

You will be able to do this w/ glEnable/PopMatrix, glRotatef, glTranslatef, and glScalef.
Contents of project2B.cxx

- Routine for generating spheres
- Routine for generating cylinders
- Routine for generating head, eyes, and pupils

- We will studied these together in class last time.
What is the correct answer?

- The correct answer is:
  - Something that looks like a dog
    - No obvious problems with output geometry.
  - Something that uses the sphere and cylinder classes.
    - If you use something else, please clear it with me first.
      - I may fail your project if I think you are using outside resources that make the project too easy.
  - Something that uses rotation for the neck and tail.

- Aside from that, feel free to be as creative as you want ... color, breed, etc.
For your reference: my dog
Outline

- Review
- Project 2B
- CW/CCW
- Geometry Creation
- Midterm Review
What is going on here?

Details:
Diffuse = 0.8, ambient = 0.2
Default GL normal is (0,0,1)
But wait...

If you have an open surface, then there is a “back face”. The back face has the opposite normal.

How can we deal with this case?

Idea #1: encode all triangles twice, with different normals
Idea #2: modify diffuse lighting model

Diffuse light = abs(L·N)

This is called two-sided lighting
Reminder: open surface, closed surface

- Closed surface:
  - you could drop it in water and it would float (i.e., no way for water to get inside)
  - no way to see the inside

- Open surface:
  - water can get to any part of the surface
  - you can see the inside
Front face / back face

- Front face: the face that is “facing outward”
- Back face: the face that is “facing inward”
- These distinctions are meaningful for closed surfaces
- They are not meaningful for open surfaces
  - This is why we did “two-sided lighting”
What is going on here?

Answer:
OpenGL is giving us diffuse+ambient on the left, but only ambient on the right (rather, diffuse == 0). Why? … it is interpreting the left side as a “front face” and the right side as a “back face”.
(if we spun the camera, the left would be gray, and the right would be white)
Determining Front Face and Back Face

- The front face and back face is determined by convention

- Convention #1:

  V1, V2, and V3 are arranged counter-clockwise around the front-face
Determining Front Face and Back Face

- The front face and back face is determined by convention
- Convention #1:

Which vertex is specified first doesn’t matter, as long as they are arranged counter-clockwise around the front-face
The front face and back face is determined by convention.

Convention #2:

V1, V2, and V3 are arranged clockwise around the front-face.
The front face and back face is determined by convention.

- **Convention #1:**
  - Front-face
  - Back-face

- **Convention #2:**
  - Front-face
  - Back-face
glFrontFace — define front- and back-facing polygons

**Name**

glFrontFace — define front- and back-facing polygons

**C Specification**

```c
void glFrontFace(GLenum mode);
```

**Python Specification**

```python
glFrontFace(mode) → None
```

**Parameters**

`mode`

Specifies the orientation of front-facing polygons. `GL_CW` and `GL_CCW` are accepted. The initial value is `GL_CCW`.

**Description**

In a scene composed entirely of opaque closed surfaces, back-facing polygons are never visible. Eliminating these invisible polygons has the obvious benefit of speeding up the rendering of the image. To enable and disable elimination of back-facing polygons, call `glEnable` and `glDisable` with argument `GL_CULL_FACE`.

The projection of a polygon to window coordinates is said to have clockwise winding if an imaginary object following the path from its first vertex, its second vertex, and so on, to its last vertex, and finally back to its first vertex, moves in a clockwise direction about the interior of the polygon. The polygon's winding is said to be counterclockwise if the imaginary object following the same path moves in a counterclockwise direction about the interior of the polygon. `glFrontFace` specifies whether polygons with clockwise winding in window coordinates, or counterclockwise winding in window coordinates, are taken to be front-facing. Passing `GL_CCW` to `mode` selects counterclockwise polygons as front-facing; `GL_CW` selects clockwise polygons as front-facing. By default, counterclockwise polygons are taken to be front-facing.
Why front face / back face is important

- **Reason #1:**
  - Need to know this if you are doing lighting calculations

- **Reason #2:**
  - Culling
Culling

- Idea: some triangles can’t affect the picture, so don’t render them.
- Question: how can this happen?
- Answer #1: geometry outside [0->width, 0->height] in device space
- Answer #2: closed surface, and the back face is facing the camera
glCullFace — specify whether front- or back-facing facets can be culled

C Specification

void glCullFace(GLenum mode);

Parameters

mode

Specifies whether front- or back-facing facets are candidates for culling. Symbolic constants GL_FRONT, GL_BACK, and GL_FRONT_AND_BACK are accepted. The initial value is GL_BACK.

Description

glCullFace specifies whether front- or back-facing facets are culled (as specified by mode) when facet culling is enabled. Facet culling is initially disabled. To enable and disable facet culling, call the glEnable and glDisable commands with the argument GL_CULL_FACE. Facets include triangles, quadrilaterals, polygons, and rectangles.

glFrontFace specifies which of the clockwise and counterclockwise facets are front-facing and back-facing. See glFrontFace.
Something you should know about:
Left- and right-handed coordinates

OpenGL: right-handed
DirectX: left-handed
Outline

- Review
- Project 2B
- CW/CCW
- Geometry Creation
- Midterm Review
Code

- Look at project 2B code
Motion Capture: Kinect

RGB+depth at 640x480, 30 fps

Kinect 2: 1080p, 2GB/s
Motion Capture

- Motion Capture basically means recording movement and translating that movement/information into another medium.
  - Often used in films and video games.
- In present day, motion capture is primarily seen as the method of recording an actor’s or multiple actors’ performance and using the information recorded to create a digital performance with a model in 3D.
In this method actors are recorded with multiple cameras from various angles. During the performance the actor wears markers near each joint on their body so the cameras can record the complete movement of the body. The movement is then pinpointed to just the markers.

Once the motion capture information is collected and the 3D model is created the two are then paired creating life-like movement.
Animation Software: Maya

Overview

Comprehensive 3D animation software

Autodesk® Maya® 3D animation software offers a comprehensive creative feature set for 3D computer animation, modeling, simulation, rendering, and compositing on a highly extensible production platform. Maya now has next-generation display technology, accelerated modeling workflows, and new tools for handling complex data.

Maya features

- Next-gen viewport display and shading
  Harness the power of Microsoft® DirectX® 11 software with the Maya DX11 shader.
  See all features

- Accelerated modeling workflow
  Increase productivity with accelerated polygon modeling workflows.

- Scene Assembly tools for smarter data
  Create large, complex worlds more easily with new Scene Assembly tools.

Get more in a suite

Extend your animation workflow with Autodesk® Entertainment Creation Suite.
Explore Entertainment Creation Suites

https://www.youtube.com/watch?v=kyTryl1jp30
https://www.youtube.com/watch?v=kyTryl1jp30
Outline

- Review
- Project 2B
- CW/CCW
- Geometry Creation
- Midterm Review
Midterm

- We have had 7 collected projects
- The test will be ~7 questions
- Closed book
- Anyone who really understands the projects should do very well on the midterm
  - You will rasterize geometry
  - You will need to know the Phong shading equation
    - I know it is a pain to memorize, esp the reflection ray
  - You will not need to memorize the 16 entries associated with any of the transformation matrices, but you should be comfortable with using the matrices