University of British Columbia
CPSC 314 Computer Graphics
Jan-Apr 2010

Tamara Munzner

Rendering Pipeline

Week 2, Mon Jan 11

http://www.ugrad.cs.ubc.ca/~cs314/Vjan2010

News

- Labs start this week
  - correction to previous slides lab is in 005, not 011
  - Mon 2-3, Shailen
  - Tue 1-2, Kai
  - Thu 10-11, Shailen
  - Fri 12-1, Garrett
- My office hours Fri 4-5 in 005 lab
  - or by appointment in my X661 office
- Leftover handouts will be in lab
- UBC CS dept announcements

Department of Computer Science
Undergraduate Events

Events this week

Drop-In Resume Edition
Date: Mon. Jan 11
Time: 11 am – 2 pm
Location: Rm 255, ICICS/CS

Industry Panel
Speakers: Managers from IBM, Microsoft, SAP, TELUS, Radical …
Date: Tues. Jan 12
Time: Panel: 5:15 – 6:15 pm Networking: 6:15 – 7:15 pm
Location: DMP 110 for panel, X-wing ugrad lounge for networking

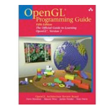
Tech Career Fair
Date: Wed. Jan 13
Time: 10 am – 4 pm
Location: SUB Ballroom

Google Tech Talk
Date: Wed, Jan 13
Time: 4 – 5 pm
Location: DMP 110

IBM Info Session
Date: Wed,, Jan 13
Time: 5:30 – 7 pm
Location: Wesbrook 100

Today's Readings

- today
  - RB Chap Introduction to OpenGL
  - RB Chap State Management and Drawing Geometric Objects
  - RB App Basics of GLUT (Aux in v 1.1)
- RB = Red Book = OpenGL Programming Guide
- http://fly.cc.fer.hr/~unreal/theredbook/



Correction: Vect-Vect Mult, The Sequel

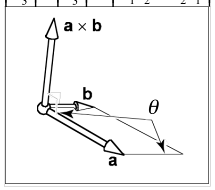
- multiply: vector * vector = vector
- cross product
  - algebraic

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \times \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} u_2v_3 - u_3v_2 \\ u_3v_1 - u_1v_3 \\ u_1v_2 - u_2v_1 \end{bmatrix}$$

  - geometric

$$\|\mathbf{a} \times \mathbf{b}\| = \boxed{\|\mathbf{a}\|\|\mathbf{b}\|}\sin\theta$$

- $\|\mathbf{a}\times\mathbf{b}\|$ parallelogram area
- $\mathbf{a}\times\mathbf{b}$ perpendicular to parallelogram



Rendering Pipeline

Rendering

- goal
  - transform computer models into images
  - may or may not be photo-realistic
- interactive rendering
  - fast, but limited quality
  - roughly follows a fixed patterns of operations
    - rendering pipeline
- offline rendering
  - ray tracing
  - global illumination

Rendering

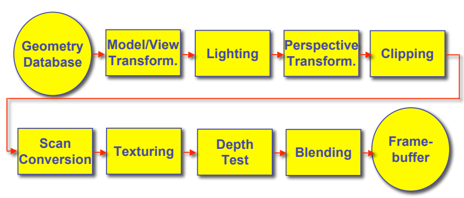
- tasks that need to be performed (in no particular order):
  - project all 3D geometry onto the image plane
    - geometric transformations
  - determine which primitives or parts of primitives are visible
    - hidden surface removal
  - determine which pixels a geometric primitive covers
    - scan conversion
  - compute the color of every visible surface point
    - lighting, shading, texture mapping

Rendering Pipeline

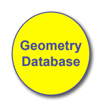
- what is the pipeline?
  - abstract model for sequence of operations to transform geometric model into digital image
  - abstraction of the way graphics hardware works
  - underlying model for application programming interfaces (APIs) that allow programming of graphics hardware
    - OpenGL
    - Direct 3D
- actual implementation details of rendering pipeline will vary

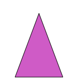
---

Rendering Pipeline



Geometry Database

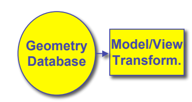


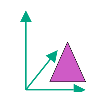


- geometry database
  - application-specific data structure for holding geometric information
  - depends on specific needs of application
    - triangle soup, points, mesh with connectivity information, curved surface

---

Model/View Transformation

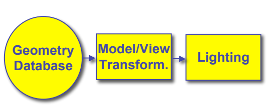


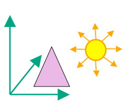


- modeling transformation
  - map all geometric objects from local coordinate system into world coordinates
- viewing transformation
  - map all geometry from world coordinates into camera coordinates

---

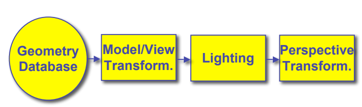
Lighting

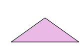




- lighting
  - compute brightness based on property of material and light position(s)
  - computation is performed *per-vertex*

Perspective Transformation

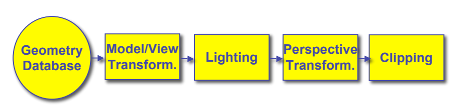


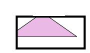


- perspective transformation
  - projecting the geometry onto the image plane
  - projective transformations and model/view transformations can all be expressed with 4x4 matrix operations

---

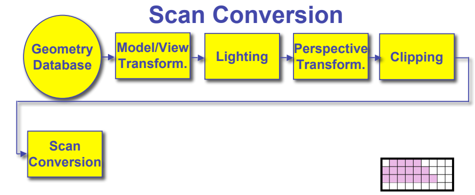
Clipping





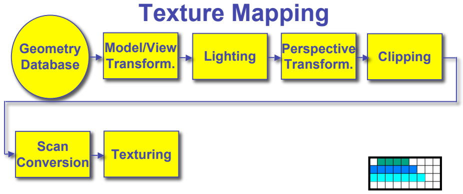
- clipping
  - removal of parts of the geometry that fall outside the visible screen or window region
  - may require *re-tessellation* of geometry

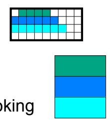
Scan Conversion



- scan conversion
  - turn 2D drawing primitives (lines, polygons etc.) into individual pixels (discretizing/sampling)
  - interpolate color across primitive
  - generate discrete fragments

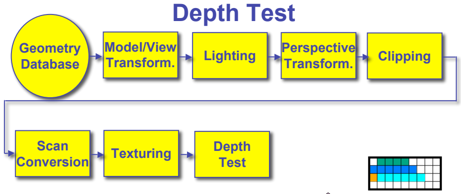
Texture Mapping

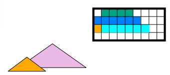




- texture mapping
  - "gluing images onto geometry"
  - color of every fragment is altered by looking up a new color value from an image

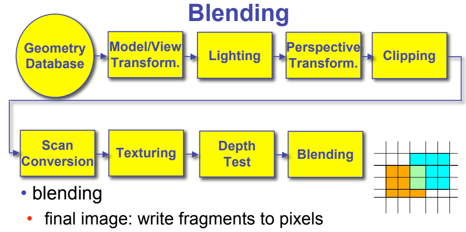
Depth Test





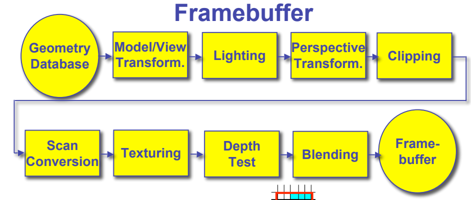
- depth test
  - remove parts of geometry hidden behind other geometric objects
  - perform on every individual fragment
    - other approaches (later)

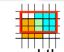
Blending



- blending
  - final image: write fragments to pixels
  - draw from farthest to nearest
  - no blending – replace previous color
  - blending: combine new & old values with arithmetic operations

Framebuffer





- framebuffer
  - video memory on graphics board that holds image
  - double-buffering: two separate buffers
    - draw into one while displaying other, then swap to avoid flicker

255 255 255	255 255 255	0 255 255	0 255 255	0 255 255
255 155 0	255 155 0	155 255 155	0 255 255	0 255 255
255 155 0	255 155 0	155 255 155	0 255 255	0 255 255

Pipeline Advantages

- modularity: logical separation of different components
- easy to parallelize
  - earlier stages can already work on new data while later stages still work with previous data
  - similar to pipelining in modern CPUs
  - but much more aggressive parallelization possible (special purpose hardware!)
  - important for hardware implementations
- only local knowledge of the scene is necessary

Pipeline Disadvantages

- limited flexibility
- some algorithms would require different ordering of pipeline stages
  - hard to achieve while still preserving compatibility
- only local knowledge of scene is available
  - shadows, global illumination difficult

OpenGL (briefly)

OpenGL

- API to graphics hardware
  - based on IRIS_GL by SGI
- designed to exploit hardware optimized for display and manipulation of 3D graphics
- implemented on many different platforms
- low level, powerful flexible
- pipeline processing
  - set state as needed

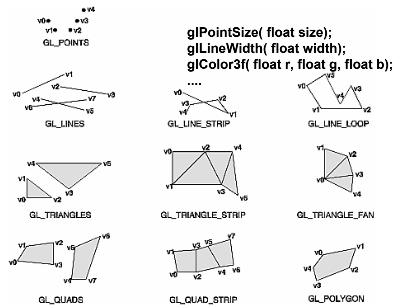
Graphics State

- set the state once, remains until overwritten
  - glColor3f(1.0, 1.0, 0.0) → set color to yellow
  - glSetClearColor(0.0, 0.0, 0.2) → dark blue bg
  - glEnable(LIGHT0) → turn on light
  - glEnable(GL_DEPTH_TEST) → hidden surf.

Geometry Pipeline

- tell it how to interpret geometry
  - glBegin(<*mode of geometric primitives*>)
  - *mode* = GL_TRIANGLE, GL_POLYGON, etc.
- feed it vertices
  - glVertex3f(-1.0, 0.0, -1.0)
  - glVertex3f(1.0, 0.0, -1.0)
  - glVertex3f(0.0, 1.0, -1.0)
- tell it you're done
  - glEnd()

Open GL: Geometric Primitives




```
glPointSize( float size);
glLineWidth( float width);
glColor3f( float r, float g, float b);
....
```


Code Sample

```
void display()
{
  glClearColor(0.0, 0.0, 0.0, 0.0);
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(0.0, 1.0, 0.0);
  glBegin(GL_POLYGON);
    glVertex3f(0.25, 0.25, -0.5);
    glVertex3f(0.75, 0.25, -0.5);
    glVertex3f(0.75, 0.75, -0.5);
    glVertex3f(0.25, 0.75, -0.5);
  glEnd();
  glFlush();
}
```

- more OpenGL as course continues

GLUT

GLUT: OpenGL Utility Toolkit

- developed by Mark Kilgard (also from SGI)
- simple, portable window manager
  - opening windows
    - handling graphics contexts
  - handling input with callbacks
    - keyboard, mouse, window reshape events
  - timing
    - idle processing, idle events
- designed for small/medium size applications
- distributed as binaries
  - free, but not open source

Event-Driven Programming

- main loop not under your control
  - vs. batch mode where you control the flow
- control flow through event callbacks
  - redraw the window now
  - key was pressed
  - mouse moved
- callback functions called from main loop when events occur
  - mouse/keyboard state setting vs. redrawing

GLUT Callback Functions

```
  // you supply these kind of functions

void reshape(int w, int h);
void keyboard(unsigned char key, int x, int y);
void mouse(int but, int state, int x, int y);
void idle();
void display();

  // register them with glut

glutReshapeFunc(reshape);
glutKeyboardFunc(keyboard);
glutMouseFunc(mouse);
glutIdleFunc(idle);
glutDisplayFunc(display);
```

```
void glutDisplayFunc (void (*func)(void));
void glutKeyboardFunc (void (*func)(unsigned char key, int x, int y));
void glutIdleFunc (void (*func)());
void glutReshapeFunc (void (*func)(int width, int height));
```

GLUT Example 1

```
#include <GLUT/glut.h>
void display()
{
  glClearColor(0,0,0,1);
  glClear(GL_COLOR_BUFFER_BIT);
  glColor4f(0,1,0,1);
  glBegin(GL_POLYGON);
  glVertex3f(0.25, 0.25, -0.5);
  glVertex3f(0.75, 0.25, -0.5);
  glVertex3f(0.75, 0.75, -0.5);
  glVertex3f(0.25, 0.75, -0.5);
  glEnd();
  glutSwapBuffers();
}
```

```
int main(int argc,char**argv)
{
 glutInit( &argc, argv );
 glutInitDisplayMode(
  GLUT_RGB|GLUT_DOUBLE);
 glutInitWindowSize(640,480);
 glutCreateWindow("glut1");
 glutDisplayFunc( display );
 glutMainLoop();
 return 0; // never reached
}
```


GLUT Example 2

```
#include <GLUT/glut.h>
void display()
{
  glRotatef(0.1, 0,0,1);

  glClearColor(0,0,0,1);
  glClear(GL_COLOR_BUFFER_BIT);
  glColor4f(0,1,0,1);
  glBegin(GL_POLYGON);
  glVertex3f(0.25, 0.25, -0.5);
  glVertex3f(0.75, 0.25, -0.5);
  glVertex3f(0.75, 0.75, -0.5);
  glVertex3f(0.25, 0.75, -0.5);
  glEnd();
  glutSwapBuffers();
}
```

```
int main(int argc,char**argv)
{
 glutInit( &argc, argv );
 glutInitDisplayMode(
  GLUT_RGB|GLUT_DOUBLE);
 glutInitWindowSize(640,480);
 glutCreateWindow("glut2");
 glutDisplayFunc( display );
 glutMainLoop();
 return 0; // never reached
}
```


Redrawing Display

- display only redrawn by explicit request
  - glutPostRedisplay() function
  - default window resize callback does this
- idle called from main loop when no user input
  - good place to request redraw
  - will call display next time through event loop
- should return control to main loop quickly
- continues to rotate even when no user action

GLUT Example 3

```
#include <GLUT/glut.h>
void display()
{
  glRotatef(0.1, 0,0,1);

  glClearColor(0,0,0,1);
  glClear(GL_COLOR_BUFFER_BIT);
  glColor4f(0,1,0,1);
  glBegin(GL_POLYGON);
  glVertex3f(0.25, 0.25, -0.5);
  glVertex3f(0.75, 0.25, -0.5);
  glVertex3f(0.75, 0.75, -0.5);
  glVertex3f(0.25, 0.75, -0.5);
  glEnd();
  glutSwapBuffers();
}
```

```
void idle() {
    glutPostRedisplay();
}

int main(int argc,char**argv)
{
 glutInit( &argc, argv );
 glutInitDisplayMode(
  GLUT_RGB|GLUT_DOUBLE);
 glutInitWindowSize(640,480);
 glutCreateWindow("glut1");
 glutDisplayFunc( display );
 glutIdleFunc( idle );
 glutMainLoop();
 return 0; // never reached
}
```


Keyboard/Mouse Callbacks

- again, do minimal work
- consider keypress that triggers animation
  - do not have loop calling display in callback!
    - what if user hits another key during animation?
  - instead, use shared/global variables to keep track of state
    - yes, OK to use globals for this!
  - then display function just uses current variable value

GLUT Example 4

```
#include <GLUT/glut.h>

bool animToggle = true;
float angle = 0.1;

void display() {
  glRotatef(angle, 0,0,1);
   ...
}
void idle() {
  glutPostRedisplay();
}
int main(int argc,char**argv)
{  ...
  glutKeyboardFunc( doKey );
   ...
}
```

```
void doKey(unsigned char key,
            int x, int y) {
  if ('t' == key) {
    animToggle = !animToggle;
    if (!animToggle)
      glutIdleFunc(NULL);
    else
      glutIdleFunc(idle);
  } else if ('r' == key) {
    angle = -angle;
  }
  glutPostRedisplay();
}
```


Readings for Next Four Lectures

- FCG Chap 6 Transformation Matrices
  - *except* 6.1.6, 6.3.1
- FCG Sect 13.3 Scene Graphs
- RB Chap Viewing
  - Viewing and Modeling Transforms *until* Viewing Transformations
  - Examples of Composing Several Transformations *through* Building an Articulated Robot Arm
- RB Appendix Homogeneous Coordinates and Transformation Matrices
  - *until* Perspective Projection
- RB Chap Display Lists