University of British Columbia
CPSC 314 Computer Graphics
Jan-Apr 2008

Tamara Munzner

Math Review
Rendering Pipeline

Week 2, Mon Jan 14

http://www.ugrad.cs.ubc.ca/~cs314/Vjan2008

News

- Tamara lecturing now!
- Labs start this week
  - Mon 12-1, Tue 1-2, Thu 10-11, Fri 12-1
- Reminder: my office hours Wed/Fri 2-3
  - in your 011 lab
  - or by appointment in my X661 office
- Leftover handouts will be in 011 lab

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/

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

Correction: Vector-Vector Multiplication

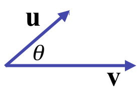
- multiply: vector * vector = scalar
- dot product, aka inner product

$$\mathbf{u} \bullet \mathbf{v}$$

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \bullet \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = (u_1 * v_1) + (u_2 * v_2) + (u_3 * v_3)$$

$$\mathbf{u} \bullet \mathbf{v} = \|\mathbf{u}\| \|\mathbf{v}\| \cos\theta$$

- geometric interpretation
  - lengths, angles
  - can find angle between two vectors

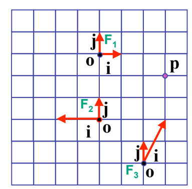


Correction: Dot Product Example

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \bullet \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = (u_1 * v_1) + (u_2 * v_2) + (u_3 * v_3)$$

$$\begin{bmatrix} 6 \\ 1 \\ 2 \end{bmatrix} \bullet \begin{bmatrix} 1 \\ 7 \\ 3 \end{bmatrix} = (6*1) + (1*7) + (2*3) = 6 + 7 + 6 = 19$$

Review: Working with Frames



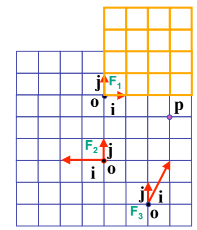
$$\mathbf{p} = \mathbf{o} + x\mathbf{i} + y\mathbf{j}$$

F_1 **p = (3,-1)**

F_2 **p = (-1.5,2)**

F_3 **p = (1,2)**

More: Working with Frames



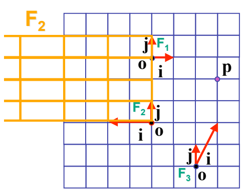
$$\mathbf{p} = \mathbf{o} + x\mathbf{i} + y\mathbf{j}$$

F_1 **p = (3,-1)**

F_2 **p = (-1.5,2)**

F_3 **p = (1,2)**

More: Working with Frames



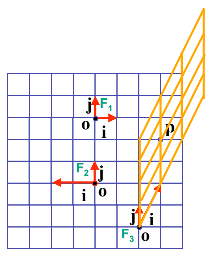
$$\mathbf{p} = \mathbf{o} + x\mathbf{i} + y\mathbf{j}$$

F_1 **p = (3,-1)**

F_2 **p = (-1.5,2)**

F_3 **p = (1,2)**

More: Working with Frames



$$\mathbf{p} = \mathbf{o} + x\mathbf{i} + y\mathbf{j}$$

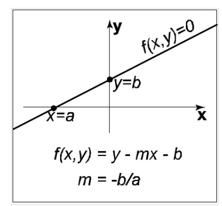
F_1 **p = (3,-1)**

F_2 **p = (-1.5,2)**

F_3 **p = (1,2)**

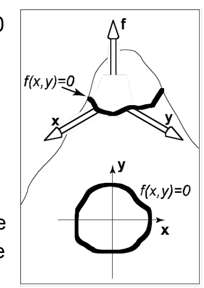
Lines

- slope-intercept form
  - y = mx + b
- implicit form
  - y – mx – b = 0
  - Ax + By + C = 0
  - f(x,y) = 0



Implicit Functions

- find where function is 0
  - plug in (x,y), check if
    - 0: on line
    - < 0: inside
    - > 0: outside
- analogy: terrain
  - sea level: f=0
  - altitude: function value
  - topo map: equal-value contours (level sets)



Implicit Circles

- $f(x,y) = (x - x_c)^2 + (y - y_c)^2 - r^2$
  - circle is points (x,y) where f(x,y) = 0
- $p = (x,y), c = (x_c, y_c) : (\mathbf{p} - \mathbf{c}) \bullet (\mathbf{p} - \mathbf{c}) - r^2 = 0$
  - points **p** on circle have property that vector from **c** to **p** dotted with itself has value r^2
- $\|\mathbf{p} - \mathbf{c}\|^2 - r^2 = 0$
  - points points **p** on the circle have property that squared distance from **c** to **p** is r^2
- $\|\mathbf{p} - \mathbf{c}\| - r = 0$
  - points **p** on circle are those a distance *r* from center point **c**

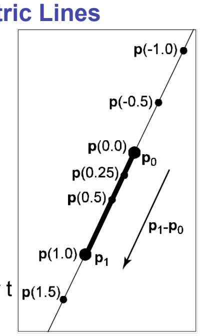
Parametric Curves

- parameter: index that changes continuously
  - (x,y): point on curve
  - t: parameter
- vector form
  - $\mathbf{p} = f(t)$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} g(t) \\ h(t) \end{bmatrix}$$

2D Parametric Lines

- $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_0 + t(x_1 - x_0) \\ y_0 + t(y_1 - y_0) \end{bmatrix}$
- $\mathbf{p}(t) = \mathbf{p}_0 + t(\mathbf{p}_1 - \mathbf{p}_0)$
- $\mathbf{p}(t) = \mathbf{o} + t(\mathbf{d})$
- start at point $\mathbf{p}_0$, go towards $\mathbf{p}_1$, according to parameter t
  - $\mathbf{p}(0) = \mathbf{p}_0$, $\mathbf{p}(1) = \mathbf{p}_1$



Linear Interpolation

- parametric line is example of general concept
  - $\mathbf{p}(t) = \mathbf{p}_0 + t(\mathbf{p}_1 - \mathbf{p}_0)$
  - interpolation
    - **p** goes through **a** at *t* = 0
    - **p** goes through **b** at *t* = 1
  - linear
    - weights *t*, (1-*t*) are linear polynomials in *t*

Matrix-Matrix Addition

- add: matrix + matrix = matrix

$$\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} + \begin{bmatrix} n_{11} & n_{12} \\ n_{21} & n_{22} \end{bmatrix} = \begin{bmatrix} n_{11}+m_{11} & n_{12}+m_{12} \\ n_{21}+m_{21} & n_{22}+m_{22} \end{bmatrix}$$

- example

$$\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} + \begin{bmatrix} -2 & 5 \\ 7 & 1 \end{bmatrix} = \begin{bmatrix} 1+(-2) & 3+5 \\ 2+7 & 4+1 \end{bmatrix} = \begin{bmatrix} -1 & 8 \\ 9 & 5 \end{bmatrix}$$

Scalar-Matrix Multiplication

- multiply: scalar * matrix = matrix

$$a\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} = \begin{bmatrix} a*m_{11} & a*m_{12} \\ a*m_{21} & a*m_{22} \end{bmatrix}$$

- example

$$3\begin{bmatrix} 2 & 4 \\ 1 & 5 \end{bmatrix} = \begin{bmatrix} 3*2 & 3*4 \\ 3*1 & 3*5 \end{bmatrix} = \begin{bmatrix} 6 & 12 \\ 3 & 15 \end{bmatrix}$$

Matrix-Matrix Multiplication

- can only multiply (n,k) by (k,m): number of left cols = number of right rows
  - legal

$$\begin{bmatrix} a & b & c \\ e & f & g \end{bmatrix}\begin{bmatrix} h & i \\ j & k \\ l & m \end{bmatrix}$$

  - undefined

$$\begin{bmatrix} a & b & c \\ e & f & g \\ o & p & q \end{bmatrix}\begin{bmatrix} h & i \\ j & k \end{bmatrix}$$

Matrix-Matrix Multiplication

- row by column

$$\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}\begin{bmatrix} n_{11} & n_{12} \\ n_{21} & n_{22} \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix}$$

$$p_{11} = m_{11}n_{11} + m_{12}n_{21}$$

Matrix-Matrix Multiplication

- row by column

$$\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}\begin{bmatrix} n_{11} & n_{12} \\ n_{21} & n_{22} \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix}$$

$$p_{11} = m_{11}n_{11} + m_{12}n_{21}$$
$$p_{21} = m_{21}n_{11} + m_{22}n_{21}$$

Matrix-Matrix Multiplication

- row by column

$$\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}\begin{bmatrix} n_{11} & n_{12} \\ n_{21} & n_{22} \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix}$$

$$p_{11} = m_{11}n_{11} + m_{12}n_{21}$$
$$p_{21} = m_{21}n_{11} + m_{22}n_{21}$$
$$p_{12} = m_{11}n_{12} + m_{12}n_{22}$$

Matrix-Matrix Multiplication

- row by column

$$\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}\begin{bmatrix} n_{11} & n_{12} \\ n_{21} & n_{22} \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix}$$

$$p_{11} = m_{11}n_{11} + m_{12}n_{21}$$
$$p_{21} = m_{21}n_{11} + m_{22}n_{21}$$
$$p_{12} = m_{11}n_{12} + m_{12}n_{22}$$
$$p_{22} = m_{21}n_{12} + m_{22}n_{22}$$

Matrix-Matrix Multiplication

- row by column

$$\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}\begin{bmatrix} n_{11} & n_{12} \\ n_{21} & n_{22} \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix}$$

$$p_{11} = m_{11}n_{11} + m_{12}n_{21}$$
$$p_{21} = m_{21}n_{11} + m_{22}n_{21}$$
$$p_{12} = m_{11}n_{12} + m_{12}n_{22}$$
$$p_{22} = m_{21}n_{12} + m_{22}n_{22}$$

- noncommutative: **AB != BA**

Matrix-Vector Multiplication

- points as column vectors: postmultiply

$$\begin{bmatrix} x' \\ y' \\ z' \\ h' \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ h \end{bmatrix} \qquad \mathbf{p'} = \mathbf{Mp}$$

- points as row vectors: premultiply

$$\begin{bmatrix} x' & y' & z' & h' \end{bmatrix} = \begin{bmatrix} x & y & z & h \end{bmatrix}\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix}^T \qquad \mathbf{p'}^T = \mathbf{p}^T\mathbf{M}^T$$

Matrices

- transpose $\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix}^T = \begin{bmatrix} m_{11} & m_{21} & m_{31} & m_{41} \\ m_{12} & m_{22} & m_{32} & m_{42} \\ m_{13} & m_{23} & m_{33} & m_{43} \\ m_{14} & m_{24} & m_{34} & m_{44} \end{bmatrix}$
- identity $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
- inverse $\mathbf{AA}^{-1} = \mathbf{I}$
  - not all matrices are invertible

Matrices and Linear Systems

- linear system of n equations, n unknowns

$$3x + 7y + 2z = 4$$
$$2x - 4y - 3z = -1$$
$$5x + 2y + z = 1$$

- matrix form **Ax=b**

$$\begin{bmatrix} 3 & 7 & 2 \\ 2 & -4 & -3 \\ 5 & 2 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 4 \\ -1 \\ 1 \end{bmatrix}$$

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

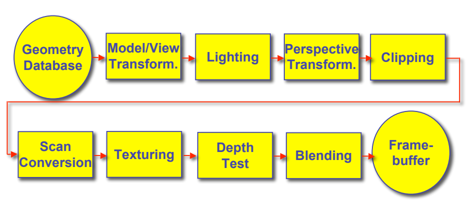
Rendering

- 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

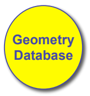
Rendering Pipeline

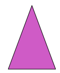
- what is the pipeline?
  - abstract model for sequence of operations to transform geometric model into digital image
  - abstraction of the way graphics hardware works
  - underlying model for application programming interfaces (APIs) that allow programming of graphics hardware
    - OpenGL
    - Direct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

Rendering Pipeline



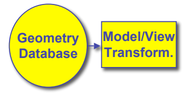
Geometry Database

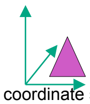




- geometry database
  - application-specific data structure for holding geometric information
  - depends on specific needs of application
    - triangle soup, points, mesh with connectivity information, curved surface

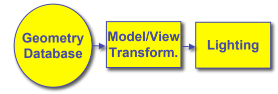
Model/View Transformation

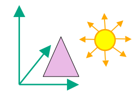




- modeling transformation
  - map all geometric objects from local coordinate system into world coordinates
- viewing transformation
  - map all geometry from world coordinates into camera coordinates

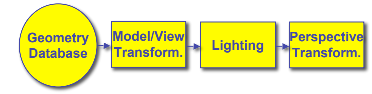
Lighting

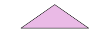




- lighting
  - compute brightness based on property of material and light position(s)
  - computation is performed *per-vertex*

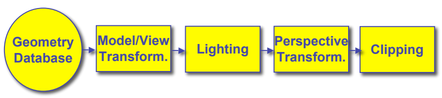
Perspective Transformation

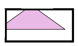




- perspective transformation
  - projecting the geometry onto the image plane
  - projective transformations and model/view transformations can all be expressed with 4x4 matrix operations

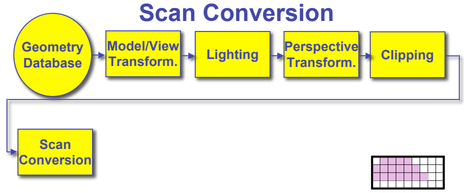
Clipping





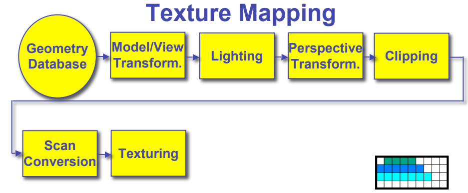
- clipping
  - removal of parts of the geometry that fall outside the visible screen or window region
  - may require *re-tessellation* of geometry

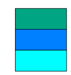
Scan Conversion



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

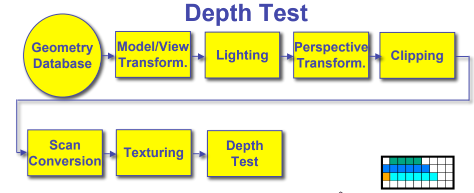
Texture Mapping

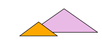




- texture mapping
  - "gluing images onto geometry"
  - color of every fragment is altered by looking up a new color value from an image

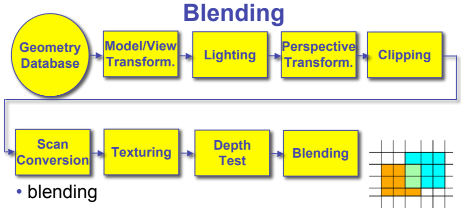
Depth Test





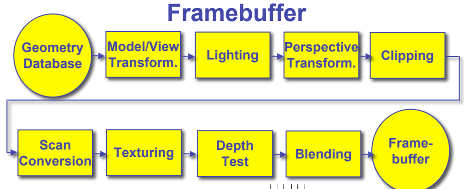
- depth test
  - remove parts of geometry hidden behind other geometric objects
  - perform on every individual fragment
    - other approaches (later)

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

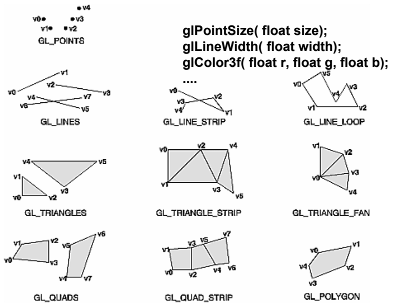
Graphics State

- 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