Intro, Math Review, OpenGL Pipeline

Week 1, Tue May 10

http://www.ugrad.cs.ubc.ca/~cs314/Vmay2005
Introduction
Expectations

- hard course!
  - heavy programming and heavy math
- fun course!
  - graphics programming addictive, create great demos
- programming prereq
  - CPSC 216 (Program Design and Data Structures)
  - course language is C++/C
- math prereq
  - MATH 200 (Calculus III)
  - MATH 221/223 (Matrix Algebra/Linear Algebra)
Course Structure

- 45% programming projects
  - 9% project 1 (building beasties with cubes and math)
  - 9% project 2 (flying)
  - 9% project 3 (shaded terrain)
  - 18% project 4 (create your own graphics game)
- 25% final
- 15% midterm (week 4, Tue 5/31)
- 15% written assignments
  - 5% each HW 1/2/3
- programming projects and homeworks synchronized
Programming Projects

- structure
  - C++, Linux
    - OK to cross-platform develop on Windows
  - OpenGL graphics library
  - GLUT for platform-independent windows/UI
  - face to face grading in lab

- Hall of Fame
  - project 1: building beasties
    - previous years: elephants, birds, poodles
  - project 4: create your own graphics game
Late Work

- 3 grace days
  - for unforeseen circumstances
  - strong recommendation: don’t use early in term
  - handing in late uses up automatically unless you tell us
- otherwise: 25% per 24 hours
  - no work accepted after solutions handed out
- exception: severe illness or crisis, as per UBC rules
  - let me know ASAP (in person or email)
  - must also turn in form with documentation

Regrading

- to request assignment or exam regrade
  - must submit detailed written explanation of why you think the grader was incorrect for the particular problem that you are disputing
- I may regrade entire assignment
  - thus even if I agree with your original request, your score may end up higher or lower
Course Information

- course web page is main resource
  - updated often, reload frequently
- newsgroup is ubc.courses.cpsc.414
  - note old course number still used
  - readable on or off campus
- (no WebCT)
Labs

- attend two labs per week, 3 sessions each
  - Tue/Thu 11-12, 3-4, 4-5
    - Thursday afternoon better than Thu morning
  - Tuesdays: example problems in spirit of written assignments and exams
  - Thursdays: help with programming projects
  - no deliverables
  - strongly recommend that you attend
Teaching Staff

- instructor: Dr. Munzner
  - tmm@cs.ubc.ca
  - office hrs in CICSR 011
    - Mon 4:30-5:30
- TAs: Warren Cheung, Greg Kempe
  - wcheung@cs.ubc.ca
  - kempe@cs.ubc.ca
- use newsgroup not email for all questions that other students might care about
Required Reading

- Fundamentals of Computer Graphics
  - Peter Shirley, AK Peters

- OpenGL Programming Guide, v 1.4
  - OpenGL Architecture Review Board
  - v 1.1 available for free online

- readings posted on schedule page
Learning OpenGL

- this is a graphics course using OpenGL
  - not a course *on* OpenGL
- upper-level class: learning APIs mostly on your own
  - only minimal lecture coverage
    - basics, some of the tricky bits
- OpenGL Red Book
- many tutorial sites on the web
  - nehe.gamedev.net
Plagiarism and Cheating

- don’t cheat, I will prosecute
  - insult to your fellow students and to me

- programming and assignment writeups must be individual work
  - exception: project 3 can be team of two
  - can discuss ideas, browse Web
  - but cannot just copy code or answers

- you must be able to explain algorithms during face-to-face demo
  - or no credit for that part of assignment, possible prosecution
Citation

- cite all sources of information
  - web sites, study group members, books
  - README for programming projects
  - end of writeup for written assignments
  - [http://www.ugrad.cs.ubc.ca/~cs314/Vmay2005/policies.html#plag](http://www.ugrad.cs.ubc.ca/~cs314/Vmay2005/policies.html#plag)
What is Computer Graphics?

- create or manipulate images with computer
- this course: algorithms for image generation
What is CG used for?

- graphical user interfaces
  - modeling systems
  - applications
- simulation & visualization
What is CG used for?

- movies
  - animation
  - special effects
What is CG used for?

- computer games
What is CG used for?

- images
  - design
  - advertising
  - art
What is CG used for?

- virtual reality / immersive displays
Real or CG?

http://www.alias.com/eng/etc/fakeorfoto/quiz.html
Real or CG?
Real or CG?
Real or CG?
This Course

- we cover
  - basic **algorithms** for
    - rendering – displaying models
    - (modeling – generating models)
    - (animation – generating motion)
  - programming in OpenGL, C++
- we do not cover
  - art/design issues
  - commercial software packages
Other Graphics Courses

- CPSC 424: Geometric Modeling
- CPSC 426: Computer Animation
- CPSC 514: Image-based Modeling and Rendering
- CPSC 526: Computer Animation
- CPSC 533A: Digital Geometry
- CPSC 533B: Animation Physics
- CPSC 533C: Information Visualization
Rendering

- creating images from models
  - geometric objects
    - lines, polygons, curves, curved surfaces
  - camera
    - pinhole camera, lens systems, orthogonal
  - shading
    - light interacting with material
- Pixar Shutterbug series
  - Williams and Siegel using Renderman, 1990
  - [www.siggraph.org/education/materials/HyperGraph/shutbug.htm](http://www.siggraph.org/education/materials/HyperGraph/shutbug.htm)
Modelling Transformation: Object Placement
Viewing Transformation: Camera Placement
Perspective Projection
Depth Cueing
Depth Clipping
Colored Wireframes
Hidden Line Removal
Hidden Surface Removal
Per-Polygon Shading
Gouraud Shading
Specular Reflection
Phong Shading
Curved Surfaces
Complex Lighting and Shading
Displacement Mapping
Reflection Mapping
Modelling

- generating models
  - lines, curves, polygons, smooth surfaces
  - digital geometry
Animation

- generating motion
  - interpolating between frames, states
Math Review
Reading

- FCG Chapter 2: Miscellaneous Math
  - except for 2.11 (covered later)
  - skim 2.2 (sets and maps), 2.3 (quadratic eqns)
  - important: 2.3 (trig), 2.4 (vectors), 2.5-6 (lines)
    2.10 (linear interpolation)
      - skip 2.5.1, 2.5.3, 2.7.1, 2.7.3, 2.8, 2.9
- FCG Chapter 4.1-4.25: Linear Algebra
  - skim 4.1 (determinants)
  - important: 4.2.1-4.2.2, 4.2.5 (matrices)
    - skip 4.2.3-4, 4.2.6-7 (matrix numerical analysis)
Textbook Errata

- list at http://www.cs.utah.edu/~shirley/fcg/errata
  - p 29, 32, 39 have potential to confuse
Notation: Scalars, Vectors, Matrices

- scalar
  - (lower case, italic)
- vector
  - (lower case, bold)
- matrix
  - (upper case, bold)

\[ a \]

\[ \mathbf{a} = \begin{bmatrix} a_1 & a_2 & \ldots & a_n \end{bmatrix} \]

\[ \mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \]
Vectors

- arrow: length and direction
  - oriented segment in nD space
- offset / displacement
  - location if given origin
Column vs. Row Vectors

- row vectors
  \[ \mathbf{a}_{\text{row}} = \begin{bmatrix} a_1 & a_2 & \ldots & a_n \end{bmatrix} \]

- column vectors
  \[ \mathbf{a}_{\text{col}} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \]

- switch back and forth with transpose
  \[ \mathbf{a}_{\text{col}}^T = \mathbf{a}_{\text{row}} \]
Vector-Vector Addition

- add: vector + vector = vector
- parallelogram rule
  - tail to head, complete the triangle

**geometric**

**algebraic**

\[
\mathbf{u} + \mathbf{v} = \begin{bmatrix}
u_1 + v_1 \\
u_2 + v_2 \\
u_3 + v_3
\end{bmatrix}
\]

examples:

\[(3,2) + (6,4) = (9,6)\]
\[(2,5,1) + (3,1,-1) = (5,6,0)\]
Vector-Vector Subtraction

- subtract: vector - vector = vector

\[ \mathbf{u} - \mathbf{v} = \begin{bmatrix} u_1 - v_1 \\ u_2 - v_2 \\ u_3 - v_3 \end{bmatrix} \]

- \((3, 2) - (6, 4) = (-3, -2)\)
- \((2, 5, 1) - (3, 1, -1) = (-1, 4, 0)\)
Vector-Vector Subtraction

- subtract: vector - vector = vector

\[ \mathbf{u} - \mathbf{v} = \begin{bmatrix} u_1 - v_1 \\ u_2 - v_2 \\ u_3 - v_3 \end{bmatrix} \]

\[
(3,2) - (6,4) = (-3,-2)
\]

\[
(2,5,1) - (3,1,-1) = (-1,4,0)
\]

argument reversal
Scalar-Vector Multiplication

- multiply: scalar $\times$ vector $= \text{vector}$
  - vector is scaled

\[ a \times u = (a \times u_1, a \times u_2, a \times u_3) \]

\[ 2 \times (3,2) = (6,4) \]
\[ .5 \times (2,5,1) = (1,2.5,.5) \]
Vector-Vector Multiplication

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

\[
\begin{bmatrix}
    u_1 \\
    u_2 \\
    u_3
\end{bmatrix}
\cdot
\begin{bmatrix}
    v_1 \\
    v_2 \\
    v_3
\end{bmatrix} = (u_1 \cdot v_1) + (u_1 \cdot v_2) + (u_3 \cdot v_3)
\]
Vector-Vector Multiplication

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

\[
\begin{bmatrix}
u_1 \\
u_2 \\
u_3 \\
\end{bmatrix} \cdot \begin{bmatrix}
v_1 \\
v_2 \\
v_3 \\
\end{bmatrix} = (u_1 \cdot v_1) + (u_1 \cdot v_2) + (u_3 \cdot v_3)
\]

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

\[
u \cdot v = \|u\| \|v\| \cos \theta
\]
Dot Product Geometry

- can find length of projection of \( u \) onto \( v \)

\[
\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\| \|\mathbf{v}\| \cos \theta
\]

\[
\|\mathbf{u}\| \cos \theta = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{v}\|}
\]

- as lines become perpendicular,

\[
\mathbf{u} \cdot \mathbf{v} \rightarrow 0
\]
Dot Product Example

\[
\begin{bmatrix}
  u_1 \\
  u_2 \\
  u_3
\end{bmatrix} \cdot \begin{bmatrix}
  v_1 \\
  v_2 \\
  v_3
\end{bmatrix} = (u_1 \cdot v_1) + (u_1 \cdot v_2) + (u_3 \cdot v_3)
\]

\[
\begin{bmatrix}
  6 \\
  1 \\
  2
\end{bmatrix} \cdot \begin{bmatrix}
  1 \\
  7 \\
  3
\end{bmatrix} = (6 \cdot 1) + (1 \cdot 7) + (2 \cdot 3) = 6 + 7 + 6 = 19
\]
Vector-Vector Multiplication, The Sequel

- multiply: \( \text{vector} \times \text{vector} = \text{vector} \)
- cross product
  - algebraic
  - geometric

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

- \( \|\mathbf{a} \times \mathbf{b}\| = \|\mathbf{u}\| \|\mathbf{v}\| \sin \theta \)
  - \( \|\mathbf{a} \times \mathbf{b}\| \) parallelogram area
  - \( \mathbf{a} \times \mathbf{b} \) perpendicular to parallelogram
RHS vs LHS Coordinate Systems

- right-handed coordinate system convention
  
  right hand rule:
  index finger $x$, second finger $y$;
  right thumb points up
  
  $z = x \times y$

- left-handed coordinate system

  left hand rule:
  index finger $x$, second finger $y$;
  left thumb points down
  
  $z = x \times y$
Basis Vectors

- take any two vectors that are **linearly independent** (nonzero and nonparallel)
- can use linear combination of these to define any other vector:

\[ \mathbf{c} = w_1 \mathbf{a} + w_2 \mathbf{b} \]
Orthonormal Basis Vectors

- if basis vectors are **orthonormal** (orthogonal (mutually perpendicular) and unit length)
  - we have Cartesian coordinate system
  - familiar Pythagorean definition of distance

Orthonormal algebraic properties:

\[ \|x\| = \|y\| = 1, \quad x \cdot y = 0 \]
Basis Vectors and Origins

- **coordinate system**: just basis vectors
  - can only specify offset: vectors
- **coordinate frame**: basis vectors and origin
  - can specify location as well as offset: points

\[ \mathbf{p} = \mathbf{o} + x\mathbf{i} + y\mathbf{j} \]
Working with Frames

\[ \mathbf{p} = \mathbf{o} + x\mathbf{i} + y\mathbf{j} \]
Working with Frames

\[ p = o + xi + yj \]

\[ F_1 \quad p = (3, -1) \]
Working with Frames

\[ p = o + xi + yj \]

\[ p = (3, -1) \]
Working with Frames

\[ p = o + xi + yj \]

\[ F_1 \quad p = (3,-1) \]

\[ F_2 \quad p = (-1.5,2) \]
Working with Frames

\[ p = o + xi + yj \]

\[ F_1 \quad p = (3,-1) \]
\[ F_2 \quad p = (-1.5,2) \]
\[ F_3 \]
Working with Frames

\[ p = o + xi + yj \]

- \( F_1 \)  \( p = (3, -1) \)
- \( F_2 \)  \( p = (-1.5, 2) \)
- \( F_3 \)  \( p = (1, 2) \)
Named Coordinate Frames

- origin and basis vectors \( \mathbf{p} = \mathbf{o} + a\mathbf{x} + b\mathbf{y} + c\mathbf{z} \)
- pick canonical frame of reference
  - then don’t have to store origin, basis vectors
  - just \( \mathbf{p} = (a, b, c) \)
- convention: Cartesian orthonormal one on previous slide
- handy to specify others as needed
  - airplane nose, looking over your shoulder, ...
  - really common ones given names in CG
    - object, world, camera, screen, ...
Lines

- slope-intercept form
  - \( y = mx + b \)

- implicit form
  - \( y - mx - b = 0 \)
  - \( Ax + By + C = 0 \)
  - \( f(x,y) = 0 \)
  - \( f(x,y) = y - mx - b \)
  - \( m = -b/a \)
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 \(f(x, y) = 0\)
- \( p = (x, y), c = (x_c, y_c) : (p - c) \cdot (p - c) - r^2 = 0 \)
  - points \(p\) on circle have property that vector from \(c\) to \(p\) dotted with itself has value \(r^2\)
- \( \|p - c\|^2 - r^2 = 0 \)
  - points points \(p\) on the circle have property that squared distance from \(c\) to \(p\) is \(r^2\)
- \( \|p - c\| - r = 0 \)
  - 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

\[ \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_0 + t(x_1 - x_0) \\ y_0 + t(y_1 - y_0) \end{bmatrix} \]

- \[ p(t) = p_0 + t(p_1 - p_0) \]
- \[ p(t) = o + t(d) \]

- start at point \( p_0 \),
go towards \( p_1 \),
according to parameter \( t \)
  - \( p(0) = p_0, \ p(1) = p_1 \)
Linear Interpolation

- parametric line is example of general concept
  - \( p(t) = p_0 + t(p_1 - p_0) \)
- 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

\[
a \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} = \begin{bmatrix} a \cdot m_{11} & a \cdot m_{12} \\ a \cdot m_{21} & a \cdot m_{22} \end{bmatrix}
\]

- example

\[
3 \begin{bmatrix} 2 & 4 \\ 1 & 5 \end{bmatrix} = \begin{bmatrix} 3 \cdot 2 & 3 \cdot 4 \\ 3 \cdot 1 & 3 \cdot 5 \end{bmatrix} = \begin{bmatrix} 6 & 12 \\ 3 & 15 \end{bmatrix}
\]
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

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

\[
p_{11} = m_{11}n_{11} + m_{12}n_{21}
\]
\[
p_{21} = m_{21}n_{11} + m_{22}n_{21}
\]
\[
p_{12} = m_{11}n_{12} + m_{12}n_{22}
\]

83
### 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: \( AB \neq BA \)
Matrix Multiplication

- can only multiply if number of left rows = number of right cols
  - 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-Vector Multiplication

- points as column vectors: postmultiply
  \[
  \begin{bmatrix}
  x' \\
  y' \\
  z' \\
  h'
  \end{bmatrix} = \begin{bmatrix}
  m_{11} & m_{12} & m_{13} & m_{14} \\
  m_{21} & m_{22} & m_{23} & m_{24} \\
  m_{31} & m_{32} & m_{33} & m_{34} \\
  m_{41} & m_{42} & m_{43} & m_{44}
  \end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  z \\
  h
  \end{bmatrix}
  \]
  \[p' = Mp\]

- points as row vectors: premultiply
  \[
  \begin{bmatrix}
  x' & y' & z' & h'
  \end{bmatrix} = \begin{bmatrix}
  m_{11} & m_{12} & m_{13} & m_{14} \\
  m_{21} & m_{22} & m_{23} & m_{24} \\
  m_{31} & m_{32} & m_{33} & m_{34} \\
  m_{41} & m_{42} & m_{43} & m_{44}
  \end{bmatrix}^T \begin{bmatrix}
  x \\
  y \\
  z \\
  h
  \end{bmatrix}
  \]
  \[p'^T = p^T M^T\]
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

\[AA^{-1} = I\]

- 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
Reading

- RB Chap. Introduction to OpenGL
- RB Chap. State Management and Drawing Geometric Objects
- RB Appendix Basics of GLUT
  - (Basics of Aux in v 1.1)
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

- actual implementation details of rendering pipeline will vary
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
texture mapping
“gluing images onto geometry”
color of every fragment is altered by looking up a new color value from an image
- depth test
  - remove parts of geometry hidden behind other geometric objects
  - perform on every individual fragment
    - other approaches (later)
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
OpenGL (briefly)
Open GL

- started in 1989 by Kurt Akeley
  - based on IRIS_GL by SGI
- API to graphics hardware
- 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_TRIANGLES`, `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);

GL_POINTS

GL_LINES

GL_LINE_STRIP

GL_LINE_LOOP

GL_TRIANGLES

GL_TRIANGLE_STRIP

GL_TRIANGLE_FAN

GL_QUADS

GL_QUAD_STRIP

GL_POLYGON
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
GLUT Draw World

```c
int main(int argc, char **argv)
{
    glutInit( &argc, argv );
    glutInitDisplayMode( GLUT_RGB |
                         GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowSize( 640, 480 );
    glutCreateWindow( "openGLDemo" );
    glutDisplayFunc( DrawWorld );
    glutIdleFunc(Idle);
    glClearColor( 1,1,1 );
    glutMainLoop();

    return 0;       // never reached
}
```
Event-Driven Programming

- main loop not under your control
  - vs. procedural
- 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));
Display Function

```c
void DrawWorld() {
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    glClear( GL_COLOR_BUFFER_BIT );
    angle += 0.05; //animation
    glRotatef(angle,0,0,1); //animation
    ... // redraw triangle in new position
    glutSwapBuffers();
}
```

- directly update value of angle variable
- so, why doesn't it spin?
- only called in response to window/input event!
Idle Function

```c
void Idle() {
    angle += 0.05;
    glutPostRedisplay();
}
```

- called from main loop when no user input
- should return control to main loop quickly
  - update value of angle variable here
  - then request redraw event from GLUT
    - draw function will be called next time through
- continues to rotate even when no user action
Keyboard/Mouse Callbacks

- do minimal work
- request redraw for display
- example: keypress triggering animation
  - do not create loop in input callback!
    - what if user hits another key during animation?
  - shared/global variables to keep track of state
  - display function acts on current variable value
Labs
Thursday Lab

- labs start Thursday
  - 11-12: morning not ideal, it’s before lecture
  - 3-4, 4-5: better, try to attend afternoon if possible

- project 0
  - make sure you can compile OpenGL/GLUT
    - useful to test home computing environment
  - template: spin around obj files
  - todo: change rotation axis
  - do not hand in, not graded
  - [http://www.ugrad.cs.ubc.ca/~cs314/Vmay2005/a0](http://www.ugrad.cs.ubc.ca/~cs314/Vmay2005/a0)

- project 1
  - transformations
  - more on Thursday after transformations lecture
Remote Graphics

- OpenGL does not work well remotely
  - very slow
- only one user can use graphics at a time
  - current X server doesn’t give priority to console, just does first come first served
  - problem: FCFS policy = confusion/chaos
- solution: console user gets priority
  - only use graphics remotely if nobody else logged on
    - with ‘who’ command, “:0” is console person
  - stop using graphics if asked by console user via email
  - or console user can reboot machine out from under you