Lecture 6

More Projections and Clipping and Introduction to *OpenGL* (Graphics Library)

Friday, February 4, 2000

William H. Hsu
Department of Computing and Information Sciences, KSU
http://www.cis.ksu.edu/~bhsu

Readings:
Sections 3.12, 6.5-6.6, Foley *et al*
Section 6.7, Hearn and Baker 2e
Chapter 2, Sections 4.9, 5.7-5.8, 7.3-7.6, Angel 2e

Lecture Outline

- Projections (Concluded)
  - Review: 5-step normalizing transformation for perspective projection (N_{per})
  - Final operation in implementing view volume: clipping
- Clipping Lines (Introduction)
  - Cohen-Sutherland algorithm
  - Cyrus-Beck / Liang-Barsky algorithm
- Clipping in 3D
  - Extending 2D line clipping algorithms to 3D objects
  - Sketch (more later): clipping in homogeneous coordinates
- Introduction to *OpenGL* (http://www.opengl.org, http://www.mesa3d.org)
  - Graphics libraries: history and design rationale
  - Specification of graphics libraries: application programmer interfaces (API)
  - Key *OpenGL* functions
- Course Projects: Overview
- Next Lecture: More *OpenGL*, Introduction to Curves

3D Projections and Clipping

- Projections (Concluded)
  - Parallel projection: cuboid view volume
  - Perspective projection: truncated pyramidal view volume (frustum)
  - Problem: how to clip?
- Clipping
  - Given: coordinates for primitives (line segments, polygons, circles, ellipses, etc.)
  - Determine: *visible components* of primitives (e.g., line segments)
  - Methods
    - Solving simultaneous equations (quick rejection: testing endpoints)
    - Solving parametric equations
  - Objectives: efficiency (e.g., fewer floating point operations)
- Clipping in 3D
  - *Some* 2D algorithms extendible to 3D
  - Specification (and implementation) of view volumes needed
- Transparent Implementation in Graphics APIs: Later Today

Normalizing Transformation for Parallel Projection

- N_{par}: Transformation (Corresponding to Stack of Primitive Matrix Ops)
- 4-Step Transformation (Section 6.5.1, FVD)
  - [1] VRP → origin
    - Translate "at point" to origin
    - Purpose: normalization for impending rotation
  - [2] Rotate (x, y, z) to (u, v, n)
    - Align VRC with WC
    - Purpose: normalize directional frame of reference according to viewer
  - [3] Shear view volume
    - Apply SH_{par}
    - Purpose: align center line of view volume with z axis (Figure 6.49, FVD)
  - [4] Translate and scale to canonical parallel cuboid
    - Nonuniform scaling according to u/v range (Equation 6.35, FVD)
    - Purpose: normalize dimensions of view volume (Equation 6.36, FVD)
- Result
  - $N_{par} = S_{par} \cdot T_{par} \cdot SH_{par} \cdot R \cdot T(-VRP)$
  - Equation 6.36, FVD)

Normalizing Transformation for Perspective Projection

- N_{per}: Transformation (Corresponding to Stack of Primitive Matrix Ops)
- 5-Step Transformation (Section 6.5.2, FVD)
  - [1] VRP → origin
    - Translate "at point" to origin
    - Purpose: normalization for impending rotation
  - [2] Rotate (x, y, z) to (u, v, n)
    - Align VRC with WC
    - Purpose: normalize directional frame of reference according to viewer
  - [3] COP → origin
    - Translate "eye" to origin
    - Purpose: normalize position of reference according to viewer
  - [4] Shear view volume
    - Apply SH_{par}
    - Purpose: align center line of view volume with z axis (Figure 6.53, FVD)
  - [5] Scale to canonical perspective frustum
    - Nonuniform scaling according to ratio of sheared-z to u/v range (Equation 6.39, FVD)
    - Purpose: normalize dimensions of view volume (Equation 6.23, FVD)

Clipping Lines

- Clipping (Sections 3.11-3.12, 6.5.3-6.5.4, FVD; Sections 7.2-7.6, Angel)
  - Problem
    - Input: coordinates for primitives
    - Output: *visible components* of primitives
  - Equational solutions: simultaneous, parametric
  - Basic primitive: clip individual points (test against rectangle bounds)
- Lines (Section 3.12, FVD; Section 7.3, Angel)
  - *Clipping line segment AB against viewing rectangle R*
  - General idea 1 (equational / regional approach)
    - Divide plane into regions about R, see whether AB can possibly intersect
    - Find intersections
  - General idea 2 (parametric approach)
    - Express line as parametric equation(s): 1 matrix or 2 scalar
    - Find intersections by plugging into parametric equation (Table 3.1, FVD)
    - Use to check clipping cases

Cohen-Sutherland Algorithm

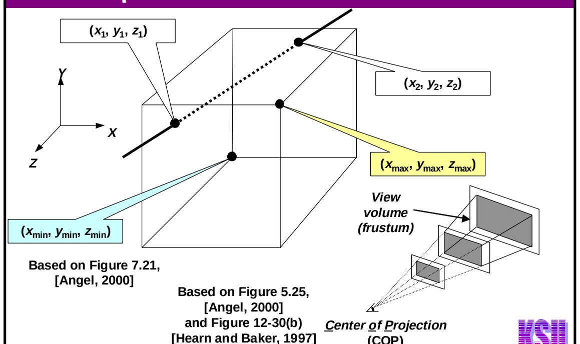
- General Idea 1 [Cohen and Sutherland, 1963]
  - Divide plane into 9 regions about and including R
  - See whether AB can possibly intersect
- Outcodes: Quick Rejection Method for Intersection Testing
  - Unique 4-bit binary number for each of 9 regions
    - b_0 = 1 iff $y > y_{max}$
    - b_1 = 1 iff $y < y_{min}$
    - b_2 = 1 iff $x > x_{max}$
    - b_3 = 1 iff $x < x_{min}$
  - Check clipping cases
    - 8 floating-point subtractions per line segment, plus integer comparison
    - Each line segment has 2 outcodes: o_1, o_2
    - Case 1: $o_1 = o_2$ = 0000 – inside; show whole segment
    - Case 2: o_1 = 0000, $o_2 \neq$ 0000 (or vice versa) – partly inside; shorten
    - Case 3: o_1 & $o_2 \neq$ 0000 – totally outside; discard
    - Case 4: o_1 & o_2 = 0000 – both endpoints outside; check further!

1001	1000	1010	y_{max}
0001	0000	0010	y_{min}
0101	0100	0110	
x_{min}	x_{max}		

Cyrus-Beck and Liang-Barsky Algorithms

- General Idea 2 [Cyrus and Beck; Liang and Barsky]
  - Express line as parametric equation(s): 1 matrix or 2 scalar
  - Find intersections by plugging into parametric equation (Table 3.1, FVD)
  - Use to check clipping cases
- Cyrus-Beck Algorithm
  - Section 3.12.4, FVD
  - More details next class (Lecture 7)
- Liang-Barsky Algorithm
  - Section 3.12.4, FVD; Section 7.3.2, Angel
  - More details next class (Lecture 7)

View Volumes in 3D: Perspective Frustum and Parallel Cuboid



Based on Figure 7.21, [Angel, 2000]

Based on Figure 5.25, [Angel, 2000] and Figure 12-30(b) [Hearn and Baker, 1997]

Generic Graphics Package: Overview

- Turn to Your Partners
  - People in your row
  - Groups numbered counterclockwise (left front to right front)
- Exercise 1 (Now): Generic Graphics Package
  - Objective: understanding generic graphics kernels
  - Exercise (5 minutes): list
    - 3 logical groups of functions that simple graphics kernels have
    - 1 criterion for deciding whether kernel function should be implemented in hardware, software, or as macro
- Exercise 2 (Later Today): Specifying Graphics Transformations
  - Objective: understanding shear transformation
  - Specification of shear transformation function
  - Implementation in *OpenGL*
- Exercise 3 (Later Today): Applying Graphics Transformations
  - Objective: using shear to implement one type of parallel projection from another
  - Enhancing capabilities of *OpenGL*

In-Class Exercises (TTYP): Generic Graphics Package

- Graphics Kernels
  - GKS
  - PHIGS (FVD)
  - *OpenGL*
- Generic Graphics Package
  - Specification
    - Requirements analysis: deciding what to include
    - Design of object model
  - Implementation
    - In hardware
    - In software (part of kernel)
    - As macros (part of kernel)
    - By application programmer

Generic Graphics Package: Typical Components

- TTYP Exercise 1a: Typical Components of Generic Graphics Kernels
  - 1. Scan conversion
  - 2. Transformations
  - 3. Clipping
  - 4. View specification / rendering
  - 5. Texturing / mapping
  - 6. 2-D primitives
  - 7. Illumination
  - 8. Color
- What Else?
  - 1. Animation
  - 2. Event handling (GUI)
  - 3. Window management

Generic Graphics Package: Specification

- TTYP Exercise 1b: Criteria for Implementation
  - In hardware
    - 1. Frequently used
    - 2. Need fast implementation
  - Library macro
    - 1. Fast
    - 2. Small, but frequently used
  - In software (library function)
    - 1. Save space (memory intensive), but not as frequently used
    - 2. Portability (possibly platform / OS dependent)
  - By applications programmer(s)
    - 1. Infrequently used but important to end-user
    - 2. Nonstandard techniques or requirements
- How Else Can We Decide At What "Level" To Place Functions?
  - 1. Cost issues: speed / frequency of use (generality of purpose) tradeoffs
  - 2. Programming language: what are non-graphical primitives?

History of Graphics Library (GL)

- Original GL (Graphics Library)
  - Developed by Silicon Graphics, Inc. (SGI)
  - Used with C under Irix (SGI Unix variant)
    - Main platforms: SGI Indigo
    - Later: SGI O2, Octane
- OpenGL Consortium
  - See [Angel, 2000] and OpenGL sites
  - Support under operating systems, IDEs (WinTel, Linux, MacOS, Amiga)
  - Linux flavor: Mesa (http://www.mesa3d.org)
    - "99% compliant" version, supported by SGI
    - Open source; licensing / validation fees not paid yet
  - Recent (last 5-8 years) adoption for academic teaching, research
- Web Resources
  - Official OpenGL web site: http://www.opengl.org
  - Porting guide, other SGI documentation: http://techpubs.sgi.com:80/library

OpenGL: Overview of Utility Toolkit (GLUT)

- Graphics Library Utility Toolkit (GLUT)
  - Chapter 2, Angel
  - Supplements and related links: http://www.aw.com/cseng
  - Links to web resources, code examples: http://www.cs.umn.edu/~angel
  - Programs from book: ftp.cs.umn.edu (pub/angel/BOOK)
  - General resources: http://www.opengl.org/Documentation/Documentation.html
- Color
  - Chapter 13, FVD; Section 2.4, Angel
  - More next month
- Viewing
  - Chapters 3 and 6, FVD; Section 2.5, Angel
  - Tutorial: http://www.eecs.tulane.edu/www/Terry/OpenGL/Introduction.html
- Window System
  - Chapter 9, FVD; Section 2.6, Angel
  - More in second half of CIS 736

OpenGL: Transformation Matrices

- *OpenGL* Matrix Stack (Section 4.9, Angel)
  - General syntax: glMatrixOperationf (parameters)
  - Loading
    - glLoadMatrixf (pointer-to-matrix)
    - Special case: glLoadIdentity ()
  - Implicit parameter: "currently loaded matrix"
    - e.g., glLoadIdentity (); glRotatef (90.0, 1.0, 0.0, 0.0); /* 90 degrees roll */
    - NB: convention – postmultiplication (glMultMatrixf)
    - Need LIFO: glPushMatrix, glPopMatrix
- Translation
  - Syntax: glTranslatef (dx, dy, dz)
- Rotation
  - Syntax: glRotatef (angle, vx, vy, vz)
  - vx, vy, vz: roll, pitch, yaw components
- Scaling
  - Syntax: glScalef (sx, sy, sz)
- Shearing: TTYP Exercise… Write glShearf (parameters)

OpenGL: Viewing API and Look-At Function

- Recall: Viewing Reference Coordinate (VRC) System Specification
  - World coordinates (x, y, z)
  - Viewing coordinates (u, v, n)
    - n ≡ view plane normal
    - v ≡ projection of VUP (view-up vector), orthogonal to n, in view plane
    - u ≡ third basis vector (orthogonal to n, v; can compute using cross product)
- Look-At Function (Section 5.2.3, Angel)
  - Syntax: gluLookAt (eyex, eyey, eyez, atx, aty, atz, upx, upy, upz)
  - eyex, eyey, eyez: specification of eyepoint e (COP *aka* view point *aka* position)
  - atx, aty, atz: specification of at point a (view reference point *aka* VRP)
  - upx, upy, upz: specification of view up vector (VUP)
- Properties of Viewing API
  - VPN = e - a
  - Specifies synthetic camera (as discussed last week)
- Now: Ready to Project…

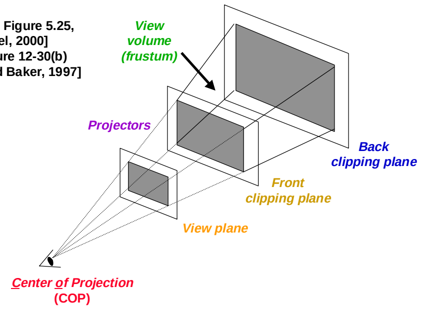
OpenGL: Orthographic and Oblique Projections

- Orthographic Projections in *OpenGL* (Section 5.7, Angel)
  - Orthographic: only parallel projections provided by *OpenGL*
  - Procedure

    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    glOrtho (-1.0, 1.0, -1.0, 1.0, -1.0, 1.0); /* canonical view volume */
  - General syntax: glOrtho (xmin, xmax, ymin, ymax, zmin ≡ near, zmax ≡ far)
- Implementing Oblique Projections
  - Problem: *OpenGL* provides only pure orthographic projections
    - Case where VPN (and projectors) || principal face normal
    - Top, front, side elevations
  - Solution
    - Q: How to implement oblique projection using glOrtho?
    - A: Use shear transformation (Chapter 6, FVD; 5.7.2 Angel… Homework 2)
    - TTYP exercise: use your glShearf to do this

The Perspective View Volume

Based on Figure 5.25, [Angel, 2000] and Figure 12-30(b) [Hearn and Baker, 1997]



Kansas State University Graphics Facilities

- KSU Graphics Infrastructure
  - Accounts
    - Computing and Information Sciences (CIS) department
    - All students should already have logins
  - Machines: KSU-CIS Beowulf cluster
  - Software: Mesa (http://www.mesa3d.org)
- Systems
  - Goodland
    - Dual boot: Windows NT 4.0, Linux
    - Matrox Millenium G400 (32Mb dual-head AGP)
    - Priority given to CIS 736 students
  - Instructional Linux systems: pending, 32Mb Pentium
  - Beowulf cluster: pending, (2) quad Pentium III Xeon-500
    - For project use only
    - Contact instructional staff to request packages

Course Project: Overview

- 3 Components
  - Project proposal (20%, 50 points)
  - Implementation (50%, 125 points)
  - Final report (30%, 75 points)
- Project Proposal (Due 02/14/2000)
  - 1-3 page description of project topic, plan
  - Guidelines: next (suggested topics, tools to appear on CIS 736 course web page)
  - See: implementation practicum links (Brown, Cornell, UNC, others) on 736 page
- Implementation
  - Students choice of programming language
  - Guidelines: next Wednesday (and on 736 page)
- Final Report
  - 4-6 page report on implementation, experimental results, interpretation
  - Peer-reviewed (does not determine grade)
  - Reviews graded (short report worth 60 points, reviews worth 15 points)

Course Project: Proposal Guidelines

- Report Contents (1-3 Pages)
  - Scope: *What kind of CG algorithms will you use?*
  - Problem: *What display problem are you addressing?*
  - Methodology: *How are you addressing the problem?*
- Scope
  - *What rendering, animation, and visualization tools (or codes) will you use?*
  - *What characteristics of the display tools are you trying to deal with / exploit?*
- Problem
  - Objective: *What is your display objective?*
  - Evaluation: *How will you demonstrate (and measure) success?*
- Methodology
  - Implementation: *What will you implement?* (general statement, not specification)
  - Graphics data representation: *How will you manipulate and represent CG data?*
  - Infrastructure: *What programming languages and platform(s) will you use?*

Terminology

- Normalizing Transformations
  - N_{par}: normalizing transformation for parallel projection (6.5.1, FVD)
  - N_{per}: normalizing transformation for perspective projection (6.5.2, FVD)
  - M: conversion matrix from perspective to parallel view volume (6.5.4, FVD)
  - $N'_{per} = M \cdot S_{per} \cdot SH_{par} \cdot T(-PRP) \cdot R \cdot T(-VRP)$ (Equation 6.49, FVD)
- Clipping: Determining Parts of Primitives to Display
  - Cohen-Sutherland: line clipping algorithm
    - Division of plane into 9 regions with (4-bit) outcodes
    - Testing endpoints of line segment
  - Parametric clipping: line / rectangle intersection using parametric equation
    - Cyrus-Beck: general convex 3D polyhedron
    - Liang-Barsky: more efficient, specialized variant (upright 2D, 3D clip regions)
- Clipping in 3D
  - Cuboid: truncated viewing pyramid used to clip after N_{par}
  - Frustum: truncated viewing pyramid
- *OpenGL*: Multiplatform, Standardized Graphics Library and API

Summary Points

- Projections: Review of N_{per}
  - [1] VRP → origin
  - [2] Rotate (x, y, z) to (u, v, n)
  - [3] COP → origin
  - [4] Shear view volume
  - [5] Scale to canonical perspective frustum
- Clipping Lines: Cohen-Sutherland, Liang-Barsky (Cyrus-Beck)
- Clipping in 3D
- Introduction to *OpenGL* (http://www.opengl.org, http://www.mesa3d.org)
  - Graphics libraries: history, design rationale, specification, APIs
  - Key *OpenGL* functions
- Course Projects: Overview
- Next Lecture
  - More *OpenGL* (Sections 10.1-10.6, Angel)
  - Intro to cubic curves (11.1, 11.2.1-11.2.2, FVD; 10.6-10.8, Hearn and Baker)