Project 1: Articulated Elephant

- modelling
  - spheres and cubes
  - hierarchical transformations
  - think cartoon!
- animation
  - more transformations
  - tail wag, head/neck nod, leg raise, trunk curl
  - gaps, self-intersections OK

News

- Project 1 out
- Trouble ticket into IT services re newsgroup on news.interchange
  - read on nnrp.cs in the meantime
Trunk Curled, Leg Raised

Interaction

• key bindings as toggles
• on click, move from rest state to new position or vice versa
• already in framework: 6 camera positions
• toggle between jumpcut and smooth transition

Transition

• first: jump cut from old to new position
  – all change happens in single frame
• do last: add smooth transition
  – change happens gradually over 30 frames
  – key click triggers animation loop
    • explicitly redraw 30 times
    • linear interpolation:
      each time, param += (new-old)/30
  – example: 5-frame transition

Tail Wag Frame 0

Tail Wag Frame 1

Tail Wag Frame 2
Strategy

- check from all camera angles
- interleave modelling, animation
  - add body part, then animate it
  - discover if on wrong track sooner
- dependencies: can’t get anim credit if no model
- do smooth transitions last
- don’t start extra credit until required all done
- consider using different model, anim xforms

Writeup

- README
  - what’s implemented
  - undone: for partial credit
    - state problems
    - describe how far you got
    - conjecture possible solutions
  - extra
    - what you did
    - how many points you argue it’s worth

Grading

- Project 1: 10% of course grade
- use handin program before Thu 2 Oct 5pm
- face-to-face grading
  - sign up for 10-minute slot, arrive 10 min early
  - bring printouts: code, README
    - must match handin
  - demo from submission directory
  - late if handin or file timestamps after deadline
    - late policy: 3 grace days for term, then 20% per day
Plagarism Policy
- no collaboration allowed
  - your work alone
  - general discussions of approach OK
  - do not look at (or copy) anybody else's code
- plagiarism is detectable
  - both by TAs and automated programs

Hall of Fame
- best work posted on course web site
- previous years

Display List recap
- reuse block of OpenGL code
- more efficient than immediate mode
  - code reuse, driver optimization
- good for static objects redrawn often
  - can’t change contents
  - not just for multiple instances
    - interactive graphics: objects redrawn every frame

Display List recap
- example: 36 snowmen
  - small display list with 36x reuse
    - 3x faster
  - big display list with 1x reuse
    - 2x faster
  - nested display lists, 1x * 36x reuse:
    - 3x faster, high-level block available

Double Buffering recap
- two framebuffers, front and back
  - avoid flicker
  - while front is on display, draw into back
  - when drawing finished, swap the two
Viewing and Projection

- need to get from 3D world to 2D image
- projection: geometric abstraction
  - what eyes or cameras do
- two pieces
  - viewing transform:
    - where is the camera, what is it pointing at?
  - perspective transform: 3D -> 2D
  - flatten to image

From Geometry to Screen

- geometry in world coordinate system: how to get to screen?
  - transform to camera coordinate system
  - transform to volume in viewing coordinates
  - clip
  - project to display coordinates
  - rasterize

Coordinate Systems

- result of a transformation
- names
  - convenience
    - elephant: neck, head, tail
  - standard conventions in graphics pipeline
  - object/modelling
  - world
  - camera/viewing
  - screen/window
  - raster/device

Projective Rendering Pipeline

- object OCS - world WCS - viewing VCS
- object modeling transformation
- world viewing transformation
- viewing projection transformation
- clipping CCS
- OCS: object coordinate system
- WCS: world coordinate system
- VCS: viewing coordinate system
- CCS: clipping coordinate system
- NDCS: normalized device coordinate system
- DCS: device coordinate system

Basic Viewing

- starting spot - OpenGL
  - camera at world origin
    - probably inside an object
    - y axis is up
    - looking down negative z axis
      - why? RHS with x horizontal, y vertical
  - translate backward so scene is visible
    - move distance d = focal length
  - what about flying around?

Arbitrary Viewing Position

- rotate/translate/scale not intuitive
- convenient formulation
  - eye point, gaze/lookat direction, up vector
Viewing Transformation

- translate eye to origin
- rotate view vector (lookat – eye) to z axis
- rotate around z to bring up into yz-plane

\[ M_{\text{cam}} \]

---

Viewing Transformation

- OpenGL
  - `gluLookAt(ex, ey, ez, lx, ly, lz, ux, uy, uz)`

but this postmultiplies the current matrix; therefore usually use as follows:

```c
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(ex, ey, ez, lx, ly, lz, ux, uy, uz);
// now ok to do model transformations
```

---

Projection

- theoretical pinhole camera
  - image inverted, more convenient equivalent

---

General Projection

image plane need not be perpendicular to view plane
Real Cameras

- real pinhole camera
- aperture
- camera
- lens

(price to pay: limited depth of field)

Projection Taxonomy

- planar projections
- parallel
- oblique
- orthographic
- cabinet
- cavalier
- top, front, side
- axonometric: isometric, dimetric, trimetric

Projection Comparison

- Obliques
  - Cavalier
  - Cabinet
- Axonometrics
  - Isometrics
  - Others
- Perspectives

http://ceprofs.tamu.edu/tkramer/ENGR%20111/5.1/20

Oblique Projections

- both have true front view
  - cavalier: distance true
  - cabinet: distance half

Axonometric Projections

- 3 equal axes
- 2 equal angles
- 0 equal angles

http://ceprofs.tamu.edu/tkramer/ENGR%20111/5.1/20

Perspective Projections

- classified by vanishing points

http://ceprofs.tamu.edu/tkramer/ENGR%20111/5.1/20
Projective Transformations

- planar geometric projections
  - planar: onto a plane
  - geometric: using straight lines
  - projections: 3D -> 2D
- aka projective mappings
- counterexamples?

Projective Transformations

- properties
  - lines mapped to lines and triangles to triangles
  - parallel lines do NOT remain parallel
    - e.g. rails vanishing at infinity
  - affine combinations are NOT preserved
    - e.g. center of a line does not map to center of projected line (perspective foreshortening)

Perspective Projection

- project all geometry through a common center of projection (eye point) onto an image plane

View Volume

- convention
  - viewing frustum mapped to specific parallelepiped
    - Normalized Device Coordinates (NDC)
    - only objects inside the parallelepiped get rendered
    - which parallelepiped? depends on rendering system
- OpenGL
  - left/right image boundaries mapped to $x = +/- 1$
  - top/bottom mapped to $y = +/- 1$
  - near/far plane mapped to $z = 0, z = 1$
Projective Transformations

- OpenGL convention
  - Camera coordinates
  - NDC

Frustum

Projective Transformations

- why near and far plane?
  - near plane:
    - avoid singularity (division by zero, or very small numbers)
  - far plane:
    - store depth in fixed-point representation (integer), thus have to have fixed range of values (0...1)
    - avoid/reduce numerical precision artifacts for distant objects

Asymmetric Frusta

- our formulation allows asymmetry
  - why bother?

Simple Formulation

- look through window center
  - symmetric frustum
  - left, right, bottom, top, near, far
    - overkill
    - nonintuitive
  - constraints
    - left = -right, bottom = -top

Field-of-View Formulation

- FOV in one direction + aspect ratio (w/h)
  - determines FOV in other direction
  - also set near, far (reasonably intuitive)

Basic Projection

- similar triangles:
  - \[ \frac{y}{d} = \frac{y}{z} \]
  - \[ \frac{y \cdot d}{z} \]
  - similarly
  - \[ \frac{x}{d} \]
Basic Projection

• using w and 4x4 matrices

\[
\begin{pmatrix}
  x \\
  y \\
  z \\
  z/d \\
\end{pmatrix}
\begin{pmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & -1 & 1 \\
\end{pmatrix}
\begin{pmatrix}
  x \\
  y \\
  z \\
  d \\
\end{pmatrix}
\]

Projective Transformations

• can express as homogeneous 4x4 matrices!
• 16 matrix entries
  – multiples of the same matrix all describe the same transformation
  – 15 degrees of freedom
  – mapping of 5 points uniquely determines transformation

Projective Transformations

• determining the matrix representation
  – need to observe 5 points in general position, e.g.
    • \([\text{left}, 0, 0, 1]^T \rightarrow [1, 0, 0, 1]^T\]
    • \([0, \text{top}, 0, 1]^T \rightarrow [0, 1, 0, 1]^T\]
    • \([0, 0, -f, 1]^T \rightarrow [0, 0, 1, 1]^T\]
    • \([0, 0, -n, 1]^T \rightarrow [0, 0, 0, 1]^T\]
    • \([\text{left}*f/n, \text{top}*f/n, -f, 1]^T \rightarrow [1, 1, 1, 1]^T\]
  – Solve resulting equation system to obtain matrix

Perspective Projection

• example
  – Assume image plane at \(z = -1\)
  – A point \([x, y, z, 1]^T\) projects to \([-x/z, -y/z, -z/z, 1]^T\) to \([x, y, z, -z]^T\)

Perspective Projection

\[
\begin{pmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & -1 & 1 \\
\end{pmatrix}
\begin{pmatrix}
  x \\
  y \\
  z \\
  -z \\
\end{pmatrix}
\]