Overview

- Background Review (first half)
  - Linear Algebra Review
- OpenGL Pipeline (second half)
- Conclusions
Vector Spaces and Linear Algebra Review

- Vector Spaces
- Vector algebra
- Useful Calculations
- More examples
- References


Vector Space

- Defn. A set whose elements are vectors with two operations 1) addition 2) multiplication by reals.
- Operations have certain properties
  - 1) addition must be commutative
  - 2) addition must be associative
  - 3) addition must have an identity element \( \vec{v} + \vec{0} = \vec{v} \)
  - 4) addition must have an inverse \( \vec{v} + \vec{w} = \vec{0} \)
Vector space cont.

- Scalar multiplication must satisfy rules
  
  \[(\alpha \beta)\vec{v} = \alpha (\beta \vec{v})\]

  \[1\vec{v} = \vec{v}\]

  \[(\alpha + \beta)\vec{v} = \alpha \vec{v} + \beta \vec{v}\]

  \[\alpha(\vec{v} + \vec{w}) = \alpha \vec{v} + \alpha \vec{w}\]

- Parallelogram rule
  - place head to tail

Vector space cont.

- Linear combination

  \[\vec{u} = \alpha_1 \vec{v}_1 + \alpha_2 \vec{v}_2 + \cdots + \alpha_n \vec{v}_n\]

- span defined as all possible linear combinations of a set

- Example: Line defined as linear combination
Affine space

- Defn. set in which geometric operations make sense, but there is not distinguished point (no origin).
- Allows for Coordinate free geometry, for example DeRose notes on web site.
- Do geometric math with points and vectors
  - point \( P \)
  - vector \( \vec{v} \)
  - calculation of a new point \( Q = P + \vec{v} \)

Affine space cont.

- Affine combination, points plus vectors \( P' = P + t(Q - P) \)
- Can be reworked to look like vector space
- Difference of points is vector
- Parametrized equation of line \( P + t(Q - P) = (1 - t)P + tQ \)
- Parametrized equation of plane \( P + t(Q - P) + s(R - P + t(Q - P)) \)

Diagram:
- Points on the line
- Point on the plane
Affine space cont.

- How are points and vectors represented in Affine space?
  \[
  \begin{bmatrix}
  x_1 \\
  y_1 \\
  z_1 \\
  1
  \end{bmatrix}
  -
  \begin{bmatrix}
  x_2 \\
  y_2 \\
  z_2 \\
  1
  \end{bmatrix}
  =
  \begin{bmatrix}
  a \\
  b \\
  c \\
  0
  \end{bmatrix}
  \]
  \[P_1 - P_2 = \vec{v}\]

- You must be careful that operations are meaningful. A point is not a vector

Affine space cont.

- Homogeneous coordinates is affine space representation, example affine 3-space in \(\mathbb{R}^4\)
- Affine 2-space in \(\mathbb{R}^3\)

- Points stay in plane
- Can homogenize any point

\[\vec{v} = P_1 - P_2\]
Homogenization Example

- Arbitrary point, 3rd coordinate will be 1

\[
\begin{bmatrix}
\frac{x}{h} \\
\frac{y}{h} \\
1
\end{bmatrix}
\rightarrow
\begin{bmatrix}
x \\
y \\
h
\end{bmatrix}
\]

Essential operations to know

- Vector operations
  - Dot product
  - Cross product
  - normalization
  - magnitude
- Euclidean operations
  - Distance between points
  - Distance from point to line
- Matrix operations
  - multiplication
  - inversion
  - determinants
  - transpose
- Linear and Affine transformations
Vector operations

- Dot product, (also known as inner product or scalar product)
  \[ \mathbf{v} \cdot \mathbf{w} = \mathbf{w} \cdot \mathbf{v} = \|\mathbf{v}\| \|\mathbf{w}\| \cos \theta \]
  (also can be calculated using the formula)
  \[ \|\mathbf{v}\| = \sqrt{\mathbf{v} \cdot \mathbf{v}} = \sqrt{x^2 + y^2 + z^2} \]

\[ \mathbf{v} \cdot \mathbf{w} = v_1 w_1 + v_2 w_2 + v_3 w_3 \]

- Recall vector magnitude is
  \[ \|\mathbf{v}\| = \sqrt{\mathbf{v} \cdot \mathbf{v}} = \sqrt{x^2 + y^2 + z^2} \]

Vector operations cont.

- Cross product (also known as vector product)

\[ \mathbf{v} \times \mathbf{w} = \text{det} \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{vmatrix} = \begin{bmatrix} v_2 w_3 - v_3 w_2 \\ v_3 w_1 - v_1 w_3 \\ v_1 w_2 - v_2 w_1 \end{bmatrix} \]
Essential operations

- Euclidean
- Matrix
- Geometric
- Affine and Linear transforms
  - postmultiply point \( p' = Tp \)
  - coordinate and geometry only notation

\[
\begin{bmatrix}
  x' \\
y' \\
1
\end{bmatrix} =
\begin{bmatrix}
a_{11} & a_{12} & t_x \\
a_{21} & a_{22} & t_y \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
1
\end{bmatrix}
\]

OpenGL Pipeline

- Basics and example program
- Vertex transformation in detail
  - OpenGL Specification, (version 1.2)
- Camera models
  - See Watt and Watt, chapter 1.
Geometry Processing in Graphics Pipeline

- db traversal
- modeling transform
- trivially accept/reject
- Lighting (for Gouraud)
- viewing transformation
- clipping
- divide by w
- map to 3D viewport
- rasterization (lighting)

Vertex Transformation Sequence
(Fig 2.6 spec, Figure 3-2 OGL Prog)
Example OpenGL program

- Cube.c, page 98
- reshape(int w, int h)
  
  ```c
  {glViewport(0,0,(Glsizei) w, Glsizei)h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity();
  glFrustum(-1.0,1.0,-1.0,1.0,1.5,20.0);
  glMatrixMode (GL_MODELVIEW);
  }
  
  display(void) {
  ...glLoadIdentity();
  gluLookAt(0.0,0.0,5.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0);
  glScalef(1.0,2.0,1.0);
  glutWireCube(1.0);
  ...
  }
  ```

Modelling Transform

- Affine transform on each part
- transform vertices

\[
\begin{bmatrix}
  a_{11} & a_{12} & a_{13} & t_x \\
  a_{21} & a_{22} & a_{23} & t_y \\
  a_{31} & a_{32} & a_{33} & t_z \\
  0 & 0 & 0 & 1
\end{bmatrix}
\]

wheel

translate's create different tires
OpenGL Camera Model

- Where camera points by default
- Points down negative z axis, from origin

```
glTranslatef(0.0, 0.0, -5.0)
```

- Can see either camera moving, or object moving, it only modifies the Model-view matrix

[-5.0]
View Transform

- Transformed to Normalized projection coordinates
- Concatenate affine transformations from world space to screen space
- floating point calculations

\[
\begin{bmatrix}
    x_e \\
    y_e \\
    z_e \\
    1
\end{bmatrix} =
\begin{bmatrix}
    a_{11} & a_{12} & a_{13} & t_x \\
    a_{21} & a_{22} & a_{23} & t_y \\
    a_{31} & a_{32} & a_{33} & t_z \\
    0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x_o \\
    y_o \\
    z_o \\
    1
\end{bmatrix}
\]

OpenGL Transformations

- Vertex’s eye coordinates from object coordinates, through postmultiplying model-view matrix by the point

\[
\begin{bmatrix}
    x_e \\
    y_e \\
    z_e \\
    w_e
\end{bmatrix} =
M
\begin{bmatrix}
    x_o \\
    y_o \\
    z_o \\
    1
\end{bmatrix}
\]
Trivially Accept/Reject

- Eliminate primitives outside of window
- Eliminate back facing primitives
- Done in eye space

Clipping

- Need to clip some primitives
OpenGL Transformations

- Vertex’s clip coordinates from eye coordinates, through postmultiplying projection matrix by the point
  \[
  \begin{bmatrix}
  x_c \\
  y_c \\
  z_c \\
  w_c
  \end{bmatrix} = P
  \begin{bmatrix}
  x_e \\
  y_e \\
  z_e \\
  w_e
  \end{bmatrix}
  \]

OpenGL Transformations cont.

- Clipping, primitives are clipped to the view volume
  \[-w_c \leq x_c \leq w_c\]
  \[-w_c \leq y_c \leq w_c\]
  \[-w_c \leq z_c \leq w_c\]

- So, the non-homogeneous coordinate is important for clipping
Perspective projection

- Perspective foreshortening, divisions
- Example projection matrix from glFrustum call

\[
\begin{bmatrix}
  x_c \\
  y_c \\
  z_c \\
  w_c
\end{bmatrix} = \begin{bmatrix}
  2n(t-l) & 0 & \frac{r+l}{r-l} & 0 \\
  0 & 2n(t-b) & \frac{t+b}{t-b} & 0 \\
  0 & 0 & \frac{f+n}{f-n} & 2fn \\
  0 & 0 & \frac{f-n}{f-n} & w_c
\end{bmatrix} \begin{bmatrix}
  x_c \\
  y_c \\
  z_c \\
  w_c
\end{bmatrix}
\]

OpenGL Transformations

- Vertex’s normalized device coordinates from clip coordinates, through dividing by \( w_c \)

\[
\begin{bmatrix}
  x_d \\
  y_d \\
  z_d \\
  1
\end{bmatrix} = \begin{bmatrix}
  x_c / w_c \\
  y_c / w_c \\
  z_c / w_c \\
  w_c / w_c
\end{bmatrix}
\]
OpenGL transformation operations

- Normalized device coordinates from object coordinates

\[
\begin{bmatrix}
& x_d \\
& y_d \\
& z_d \\
& 1
\end{bmatrix} = 1/wPM
\begin{bmatrix}
& x_o \\
& y_o \\
& z_o \\
& 1
\end{bmatrix}
\]

Mapping to viewport/window

- Map floating point coordinate locations to discrete pixel locations
OpenGL transformation Operations cont.

- Calculation of window coordinates

\[
\begin{bmatrix}
  x_w \\
  y_w \\
  z_w
\end{bmatrix} = \begin{bmatrix}
  (p_x/2)x_d + o_x \\
  (p_y/2)y_d + o_y \\
  [(f - n)/2]z_d + (n + f)/2
\end{bmatrix}
\]

- Window center \(o_x, o_y\)
- width \(p_x\) height \(p_y\)
- factor and offset computed with \(z\)Near \(n\) \(z\)Far \(f\)

OpenGL transformation Operations cont.

- Viewport set by `glViewport(int x, int y, sizei w, sizei h);` \(o_x = x + w/2, o_y = y + h/2\)
- Z depth factor and offset are set by `glDepthRange(campd n, clampd f)(zNear, zFar)`
  \(n = 0.0, f = 1.0\)
OpenGL Transformation Operations cont.

- Matrix modes are set with glMatrixMode(enum mode)
  - TEXTURE, MODELVIEW, COLOR, PROJECTION
- Standard affine transformations available:
  - translate, scale, rotate
- Perspective matrix functions also available:
  - Frustum, ortho
- glu utility for perspective is often easier to use
  or glulookat

Lighting

- Illumination from light to viewpoint (BRDF)
  - per vertex calculation
- Gouraud (lambertian)
- Phong (normal interpolation)
- Area lights
- materials
OpenGL transformations cont.

- Normal transformation
- For lighting, normals are transformed to eye coordinates by the inverse modelview matrix
- The normals are rescaled and normalized

\[
\begin{bmatrix}
  nx_e \\
  ny_e \\
  nz_e \\
  q_e
\end{bmatrix} = \begin{bmatrix}
  nx \\
  ny \\
  n_z \\
  q
\end{bmatrix} M^{-1}
\]

- What happens with inversion? Problems?

Scan conversion

- process of determining what pixels should be written
- geometric primitives to pixels (Fig. 3.4 pp. 73)

- antialiasing
- superposition and subpixel rendering
- shading is how lighting is evaluated
Rasterization for Polygons

- Figure 3.23 page 93

Texture Mapping

- Inverse mappings and reconstruction
- Can shortcut, and try to do texturing without perspective divides, but may have artifacts
Z-Buffering

- Random depth updating
- Depth for every pixel
- The range of depths is controllable by the OpenGL transformations (near and far values, as well as the distance from near and far to the center of projection)

Conclusions

- Linear algebra
- OpenGL pipeline in more detail
- Coordinate transformations important
- Familiarity with OpenGL
- Next time: Camera Calibration, Tsai’s paper assigned reading: