Affine Transformations and Transformation Hierarchies in OpenGL

Wolfgang Heidrich

Course News

Assignment 1

- Due January 31

Homework 1

- Exercise problems for transformations
- Discussed in labs next week

Reading

- Chapter 5

Recap: Properties of Affine Transformations

Theorem:

- The following statements are synonymous
  - *A transformation T(x) is affine, i.e.:*

$$\mathbf{x}' = T(\mathbf{x}) := \mathbf{M} \cdot \mathbf{x} + \mathbf{t},$$

$$\text{for some matrix } \mathbf{M} \text{ and vector } \mathbf{t}$$

  - *T(x) preserves affine combinations, i.e.*

$$T\left(\sum_{i=1}^{n} a_i \cdot \mathbf{x}_i\right) = \sum_{i=1}^{n} a_i \cdot T(\mathbf{x}_i), \text{ for } \sum_{i=1}^{n} a_i = 1$$

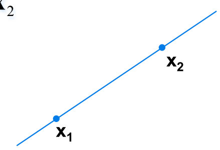
  - *T(x) maps parallel lines to parallel lines*

Recap: Properties of Affine Transformations

Example:

- Affine combination of 2 points

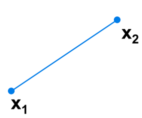
$$\begin{aligned} \mathbf{x} &= a_1 \cdot \mathbf{x}_1 + a_2 \cdot \mathbf{x}_2, \text{ with } a_1 + a_2 = 1 \\ &= (1 - a_2) \cdot \mathbf{x}_1 + a_2 \cdot \mathbf{x}_2 \\ &= \mathbf{x}_1 + a_2 \cdot (\mathbf{x}_2 - \mathbf{x}_1) \end{aligned}$$



Recap: Properties of Affine Transformations

Definition:

- A convex combination is an affine combination where all the weights a_i are positive
- Note: this implies $0 \leq a_i \leq 1$, $i=1 \ldots n$



Recap: Properties of Affine Transformations

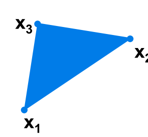
Example:

- Convex combination of 3 points

$$\mathbf{x} = \alpha \cdot \mathbf{x}_1 + \beta \cdot \mathbf{x}_2 + \gamma \cdot \mathbf{x}_3$$

$$\text{with } \alpha + \beta + \gamma = 1, \ 0 \leq \alpha, \beta, \gamma \leq 1$$

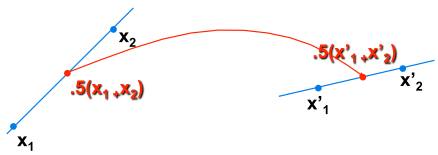
- α, β, and γ are called *Barycentric coordinates*



Recap: Properties of Affine Transformations

Preservation of affine combinations:

- Can compute transformation of every point on line or triangle by simply transforming the *control points*



Recap: Homogeneous Coordinates

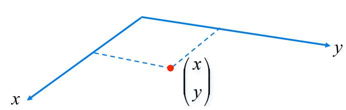
Homogeneous representation of points:

- Add an additional component $w=1$ to all *points*
- All multiples of this vector are considered to represent the same 3D point
- Use square brackets (rather than round ones) to denote homogeneous coordinates (different from text book!)

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \equiv \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \equiv \begin{bmatrix} x \cdot w \\ y \cdot w \\ z \cdot w \\ w \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix}, \forall w \neq 0$$

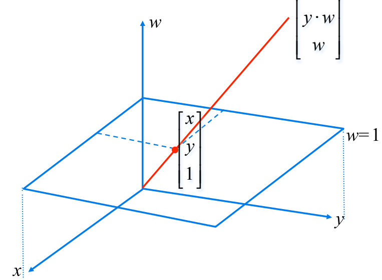
Recap: Geometrically In 2D

Cartesian Coordinates:



Recap: Geometrically In 2D

Homogeneous Coordinates: $\begin{bmatrix} x \cdot w \\ y \cdot w \\ w \end{bmatrix}$



Recap: Homogeneous Matrices

Affine Transformations

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & 0 \\ m_{2,1} & m_{2,2} & m_{2,3} & 0 \\ m_{3,1} & m_{3,2} & m_{3,3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & 0 \\ m_{2,1} & m_{2,2} & m_{2,3} & 0 \\ m_{3,1} & m_{3,2} & m_{3,3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & t_x \\ 0 & 0 & 0 & t_y \\ 0 & 0 & 0 & t_z \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Recap: Homogeneous Matrices

Combining the two matrices into one:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & 0 \\ m_{2,1} & m_{2,2} & m_{2,3} & 0 \\ m_{3,1} & m_{3,2} & m_{3,3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & t_x \\ 0 & 0 & 0 & t_y \\ 0 & 0 & 0 & t_z \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & t_x \\ m_{2,1} & m_{2,2} & m_{2,3} & t_y \\ m_{3,1} & m_{3,2} & m_{3,3} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Recap: Homogeneous Transformations

Notes:

- A composite transformation is now just the product of a few matrixes
- Rather than multiply each point sequentially with 3 matrices, first multiply the matrices, then multiply each point with only one (composite) matrix
  - *Much faster for large # of points!*
- The composite matrix describing the affine transformation always has the bottom row 0,0,1 (2D), or 0,0,0,1 (3D)

Recap: Homogeneous Matrices

Note:

- Multiplication of the matrix with a constant does not change the transformation!

$$\tilde{T}\left(\begin{bmatrix}x\\y\\z\\1\end{bmatrix}\right) = \begin{bmatrix} m_{1,1}\cdot k & m_{1,2}\cdot k & m_{1,3}\cdot k & t_x\cdot k \\ m_{2,1}\cdot k & m_{2,2}\cdot k & m_{2,3}\cdot k & t_y\cdot k \\ m_{3,1}\cdot k & m_{3,2}\cdot k & m_{3,3}\cdot k & t_z\cdot k \\ 0 & 0 & 0 & k \end{bmatrix}\cdot\begin{bmatrix}x\\y\\z\\1\end{bmatrix} = \begin{bmatrix}x'\cdot k\\y'\cdot k\\z'\cdot k\\k\end{bmatrix}$$

$$\equiv \begin{bmatrix}x'\\y'\\z'\\1\end{bmatrix} = T\left(\begin{bmatrix}x\\y\\z\\1\end{bmatrix}\right)$$

Recap: Homogeneous Vectors

Representing vectors in homogeneous coordinates

- Need representation that is only affected by linear transformations, but not by translations
- This is achieved by setting w=0

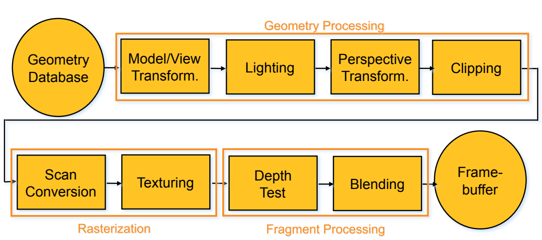
$$T\left(\begin{bmatrix}x\\y\\z\\0\end{bmatrix}\right) = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & t_x \\ m_{2,1} & m_{2,2} & m_{2,3} & t_y \\ m_{3,1} & m_{3,2} & m_{3,3} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}\cdot\begin{bmatrix}x\\y\\z\\0\end{bmatrix} = \begin{bmatrix}x'\\y'\\z'\\0\end{bmatrix}$$

Recap: Homogeneous Coordinates

Properties

- Unified representation as 4-vector (in 3D) for
  - *Points*
  - *Vectors / directions*
- Affine transformations become 4x4 matrices
  - *Composing multiple affine transformations involves simply multiplying the matrices*
  - *3D affine transformations have 12 degrees of freedom*
    - Need mapping of 4 points to uniquely define transformation

The Rendering Pipeline



Modeling Transformation

Purpose:

- Map geometry from local object coordinate system into a global world coordinate system
- Same as placing objects

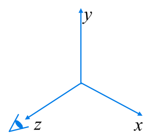
Transformations:

- Arbitrary affine transformations are possible
  - *Even more complex transformations may be desirable, but are not available in hardware*
    - Freeform deformations

Viewing Transformation

Purpose:

- Map geometry from *world coordinate system* into *camera coordinate system*
- Camera coordinate system is *right-handed*, viewing direction is *negative* z-axis
- Same a placing camera



Transformations:

- Usually only *rigid body transformations*
  - *Rotations and translations*
- Objects have same size and shape in camera and world coordinates

Model/View Transformation

Combine modeling and viewing transform.

- Combine both into a single matrix
- Saves computation time if many points are to be transformed
- Possible because the viewing transformation directly follows the modeling transformation without intermediate operations

Rendering Geometry in OpenGL

```
glBegin( GL_TRIANGLES );
  glVertex3f( x1, y1, z1 ); // vertex 1 of triangle 1
  glVertex3f( x2, y2, z2 ); // vertex 2 of triangle 1
  glVertex3f( x3, y3, z3 ); // vertex 3 of triangle 1
  glVertex3f( x4, y4, z4 ); // vertex 1 of triangle 2
  glVertex3f( x5, y5, z5 ); // vertex 2 of triangle 2
  glVertex3f( x6, y6, z6 ); // vertex 3 of triangle 2
  …
glEnd();
```


Rendering Geometry in OpenGL

Additional attributes

- glColor3f: RGB color value (0…1 per component)
- glNormal3f: normal vector
- glTexCoord2f: texture coordinate (explained later)

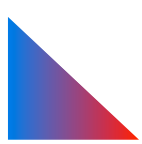
OpenGL is state machine:

- Every vertex gets color, normal etc. that corresponds to last specified value

Rendering Geometry in OpenGL

Example:

```
glBegin( GL_TRIANGLES );
  glColor3f( 1.0, 0.0, 0.0 );
  glVertex3f( 1.0, 0.0, 0.0 );
  glColor3f( 0.0, 0.0, 1.0 );
  glVertex3f( 0.0, 1.0, 0.0 );
  glVertex3f( 0.0, 0.0, 0.0 );
glEnd();
```



OpenGL Naming Scheme

Function names:

gl Vertex 3 f

- OpenGL Prefix
- Operation
- Dimensionality
  1-4
  Missing coordinates are 0 (x,y,z) or 1 (w)
- Type of parameters
  e.g. f (float)
  d (double)
  i (integer)

Matrix Operations in OpenGL

2 Matrices:

- Model/view matrix M
- Projective matrix P

Example:

```
glMatrixMode( GL_MODELVIEW );
glLoadIdentity(); // M=Id
glRotatef( angle, x, y, z ); // M=Id*R(α)
glTranslatef( x, y, z ); // M= Id*R (α)*T(x,y,z)
glMatrixMode( GL_PROJECTION );
glRotatef( … ); // P= …
```


Matrix Operations in OpenGL

Semantics:

- glMatrixMode sets the matrix that is to be affected by all following transformations (multiplication from the right)
- Transformations that affect a vertex *first* have to be specified *last*
- Whenever primitives are rendered with glBegin(), the vertices are transformed with whatever the current model/view and perspective matrix is
  - *Normals are transformed with the inverse transpose*

Matrix Operations in OpenGL

Specifying matrices (replacement)

- glLoadIdentity()
- glLoadMatrixf(GLfloat *m) // 16 floats

Specifying matrices (multiplication)

- glMultMatrixf(GLfloat *m) // 16 floats
- glRotatef(GLfloat angle, GLfloat x, Glfloat y, GLfloat z) // angle and axis
- glScalef(GLfloat x, GLfloat y, GLfloat z)
- glTranslatef(GLfloat x, GLfloat y, GLfloat z)

Matrix Operations in OpenGL

Perspective Matrices (details next lecture):

- glFrustum(left, right, bottom, top, near, far)
  - *Specifies perspective xform (near, far are always positive)*
- glOrtho(left, right, bottom, top, near, far)

Convenience Functions:

- gluPerspective(fovy, aspect, near, far)
  - *Another way to do perspective*
- gluLookAt(eyeX, eyeY, eyeZ,
centerX, centerY, centerZ,
upX, upY, upZ)
  - *Useful for viewing transform*

Interpreting Composite OpenGL Transformations

Example for earlier lectures:

- Rotation around arbitrary center
- In OpenGL:

Top-to-bottom: transf. of coordinate frame

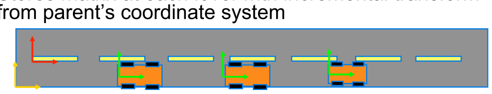
```
// initialization of matrix
glMatrixMode( GL_MODELVIEW );
glLoadIdentity();

glTranslatef( 4, 3 );
glRotatef( 30, 0.0, 0.0, 1.0 );
glTranslatef( -4, -3 );

glBegin( GL_TRIANGLES );
// specify object geometry…
```

Bottom-to-top: transf. of object

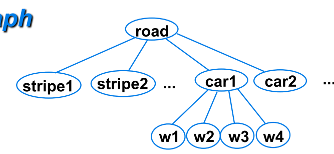
Transformation Hierarchies

Scene may have a hierarchy of coordinate systems

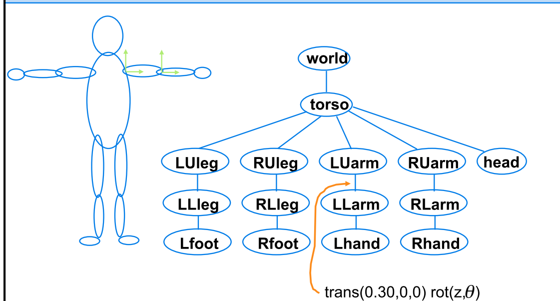
- Stores matrix at each level with incremental transform from parent's coordinate system



Scene graph

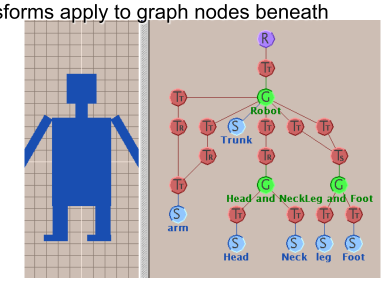


Transformation Hierarchy Example 1



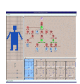
Transformation Hierarchies

- Hierarchies don't fall apart when changed
- transforms apply to graph nodes beneath



Brown Applets

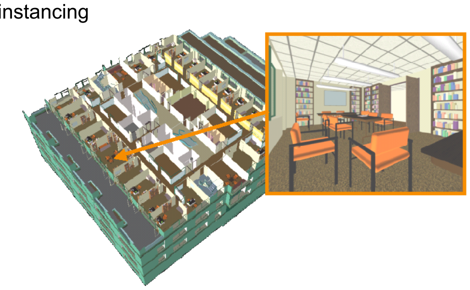
`http://www.cs.brown.edu/exploratories/freeSoftware/catalogs/scenegraphs.html`



- Have a look later

Transformation Hierarchy Example 2

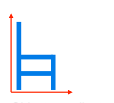
- Draw same 3D data with different transformations: instancing

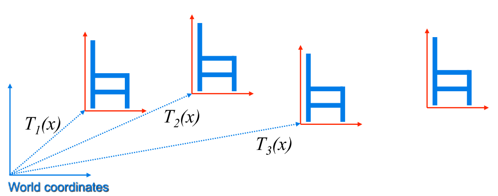


Matrix Stacks

Challenge of avoiding unnecessary computation

- Using inverse to return to origin
- Computing incremental T_1 -> T_2

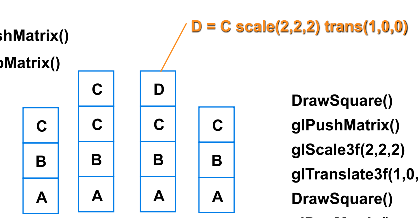




Matrix Stacks

glPushMatrix()

glPopMatrix()



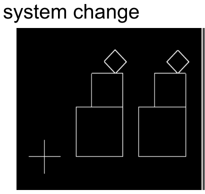

```
DrawSquare()
glPushMatrix()
glScale3f(2,2,2)
glTranslate3f(1,0,0)
DrawSquare()
glPopMatrix()
```

Modularization

Drawing a scaled square

- Push/pop ensures no coord system change

```
void drawBlock(float k) {
  glPushMatrix();

  glScalef(k,k,k);
  glBegin(GL_LINE_LOOP);
  glVertex3f(0,0,0);
  glVertex3f(1,0,0);
  glVertex3f(1,1,0);
  glVertex3f(0,1,0);
  glEnd();

  glPopMatrix();
}
```



Matrix Stacks

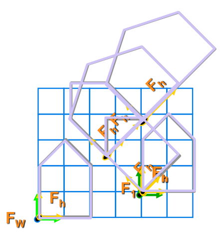
Advantages

- No need to compute inverse matrices all the time
- Modularize changes to pipeline state
- Avoids incremental changes to coordinate systems
  - *Accumulation of numerical errors*

Practical issues

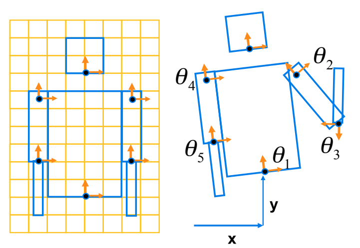
- In graphics hardware, depth of matrix stacks is limited
  - *(typically 16 for model/view and about 4 for projective matrix)*

Transformation Hierarchy Example 3




```
glLoadIdentity();
glTranslatef(4,1,0);
glPushMatrix();
glRotatef(45,0,0,1);
glTranslatef(0,2,0);
glScalef(2,1,1);
glTranslate(1,0,0);
glPopMatrix();
```


Transformation Hierarchy Example 4



glTranslate3f(x,y,0);
glRotatef(θ_1,0,0,1);
DrawBody();
glPushMatrix();
  glTranslate3f(0,7,0);
  DrawHead();
glPopMatrix();
glPushMatrix();
  glTranslate(2.5,5.5,0);
  glRotatef(θ_2,0,0,1);
  DrawUArm();
  glTranslate(0,-3.5,0);
  glRotatef(θ_3,0,0,1);
  DrawLArm();
glPopMatrix();
... (draw other arm)

Hierarchical Modeling

Advantages

- Define object once, instantiate multiple copies
- Transformation parameters often good control knobs
- Maintain structural constraints if well-designed

Limitations

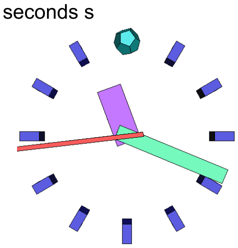
- Expressivity: not always the best controls
- Can't do closed kinematic chains
  - *Keep hand on hip*

Single Parameter: simple

Parameters as functions of other params

- Clock: control all hands with seconds s

- m = s/60, h=m/60,
- theta_s = (2 pi s) / 60,
- theta_m = (2 pi m) / 60,
- theta_h = (2 pi h) / 60



Single Parameter: complex

Mechanisms not easily expressible with affine transforms

http://www.flying-pig.co.uk

Wolfgang Heidrich

Coming Up:

Next Week:

- Perspective projection
- Lighting/shading

Wolfgang Heidrich