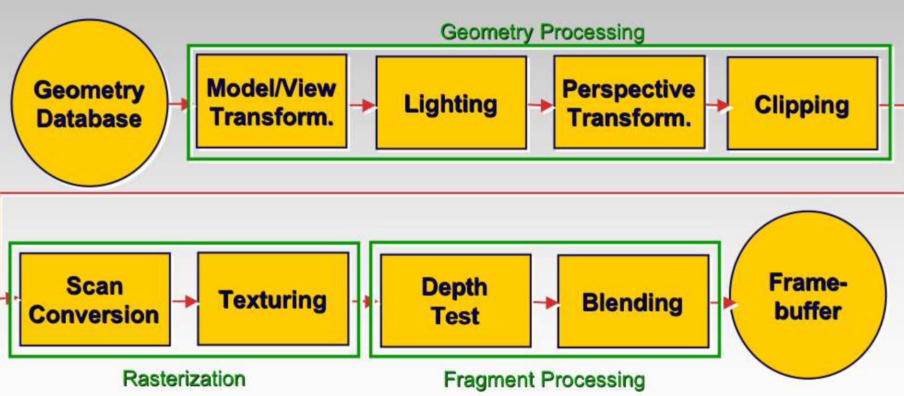
OpenGL Transformations, Hierarchical Transformations, Accelerations

CPSC 314

The Rendering Pipeline



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
- All points are represented as *column vectors*

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

Homogeneous Matrices

Affine Transformations

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

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

Homogeneous Vectors

Representing vectors in homogeneous coordinates

- Column vectors with $w=0$

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

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

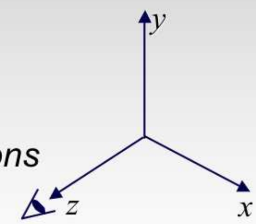
Viewing Transformation

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

Homogeneous Planes And Normals

Planes in Cartesian Coordinates:

$$\{(x, y, z)^T \mid n_x x + n_y y + n_z z + d = 0\}$$

- n_x, n_y, n_z, and d are the parameters of the plane (normal and distance from origin)

Planes in Homogeneous Coordinates:

$$\{[x, y, z, w]^T \mid n_x x + n_y y + n_z z + dw = 0\}$$

Homogeneous Planes And Normals

Planes in homogeneous coordinates are represented as row vectors

- E=$[n_x, n_y, n_z, d]$
- Condition that a point $[x,y,z,w]^T$ is located in E

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \in E = [n_x, n_y, n_z, d] \Leftrightarrow [n_x, n_y, n_z, d] \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = 0$$

Homogeneous Planes and Normals

Example in 2D (lines instead of planes):

- Line L: $y=1-x$
- Implicit definition: $-x-y+1=0$
- Unit-length normal of that line:

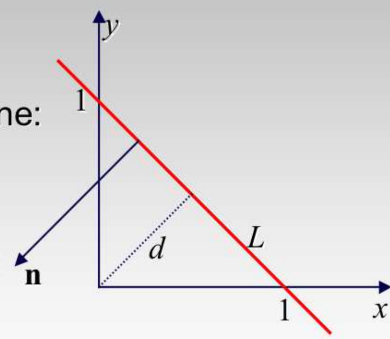
$$\mathbf{n} = \left[\frac{-\sqrt{2}}{2}, \frac{-\sqrt{2}}{2}, 0\right]^T$$

- Distance of line from origin:

$$d = \sqrt{2}/2$$

- Thus:

$$L = \left[\frac{-\sqrt{2}}{2}, \frac{-\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right] \equiv \left[-1,-1,1\right]$$

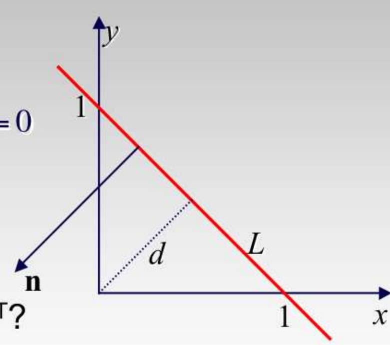


Homogeneous Planes and Normals

Example in 2D (cont.):

- Is $[1,0,1]^T$ on the line?

$$\left[\frac{-\sqrt{2}}{2}, \frac{-\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right]\begin{bmatrix}1\\0\\1\end{bmatrix} = \frac{-\sqrt{2}}{2} + 0 + \frac{\sqrt{2}}{2} = 0$$



- What about $[0,0,1]^T$, $[1,1,1]^T$?

$$\left[\frac{-\sqrt{2}}{2}, \frac{-\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right]\begin{bmatrix}0\\0\\1\end{bmatrix} = \frac{\sqrt{2}}{2}; \qquad \left[\frac{-\sqrt{2}}{2}, \frac{-\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right]\begin{bmatrix}1\\1\\1\end{bmatrix} = \frac{-\sqrt{2}}{2}$$

Homogeneous Planes And Normals

Transformations of planes

$$[n_x, n_y, n_z, d] \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = 0 \Leftrightarrow T([n_x, n_y, n_z, d]) \cdot (\mathbf{A} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}) = 0$$

Homogeneous Planes And Normals

Transformations of planes

$$[n_x, n_y, n_z, d] \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = 0 \Leftrightarrow ([n_x, n_y, n_z, d] \cdot \mathbf{A}^{-1}) \cdot (\mathbf{A} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}) = 0$$

- Works for $T([n_x, n_y, n_z, d]) = [n_x, n_y, n_z, d]\mathbf{A}^{-1}$
- Thus: planes have to be transformed by the *inverse* of the affine transformation (multiplied from left as a row vector)!

Homogeneous Planes And Normals

Homogeneous Normals

- The plane definition also contains its normal
- Normal written as a vector $[n_x, n_y, n_z, 0]^T$

$$\left(\begin{bmatrix} n_x \\ n_y \\ n_z \\ 0 \end{bmatrix} \cdot \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix}\right) = 0 \Leftrightarrow \left(\left(\mathbf{A}^{-T} \cdot \begin{bmatrix} n_x \\ n_y \\ n_z \\ 0 \end{bmatrix}\right) \cdot \left(\mathbf{A} \cdot \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix}\right)\right) = 0$$

- Thus: the normal to any surface has to be transformed by the inverse transpose of the affine transformation (multiplied from the right as a column vector)!

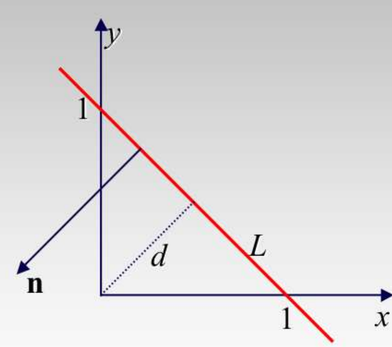
Transforming Homogeneous Normals

Back to 2D example:

- Before transformation"

$$L = \left[\frac{-\sqrt{2}}{2}, \frac{-\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right]$$

$$\mathbf{n} = \begin{bmatrix} \frac{-\sqrt{2}}{2} \\ \frac{-\sqrt{2}}{2} \\ 0 \end{bmatrix}$$

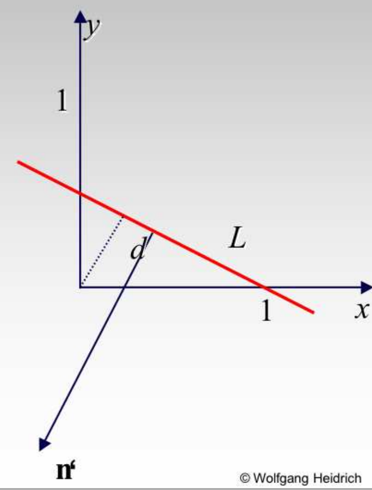


Transforming Homogeneous Normals

Scale by 1/2 in y direction

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & & \\ & 1/2 & \\ & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = M \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathbf{n}' = M^{-T}\mathbf{n} = \begin{bmatrix} 1 & & \\ & 2 & \\ & & 1 \end{bmatrix} \begin{bmatrix} \frac{-\sqrt{2}}{2} \\ \frac{-\sqrt{2}}{2} \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{-\sqrt{2}}{2} \\ -\sqrt{2} \\ 0 \end{bmatrix}$$



Transforming Homogeneous Normals

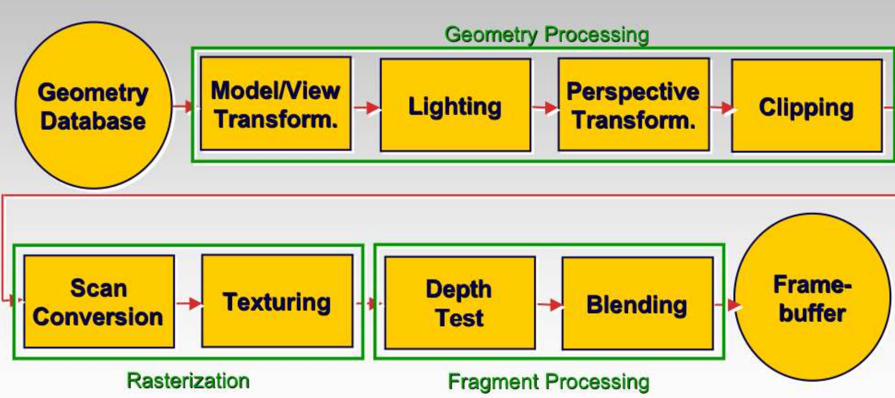
Inverse Transpose of

- Rotation by α
  - *Rotation by α*
- Scale by s
  - *Scale by 1/s*
- Translation by t
  - *Identity matrix!*
- Shear by a along x axis
  - *Shear by -a along y axis*

OpenGL Transformations, Hierarchical Transformations, Accelerations

CPSC 314

The Rendering Pipeline



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)

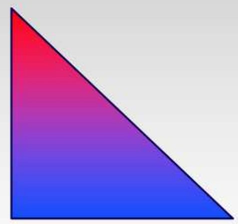
OpenGL is state machine:

- 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, 0.0, 0.0 );
    glVertex3f(1.0, 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:

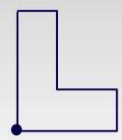
- gluPerspective(fovy, aspect, near, far)
  - *Another way to do perspective*
- gluLookAt(eyeX, eyeY, eyeZ,
  centerX, centerY, centerZ,
  upX, upY, upZ)
  - *Useful for viewing transform*

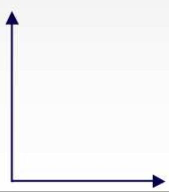
Interpreting Composite Transformations

Example for last lecture: Rotation around arbitrary center

- E.g. rotate 30 degrees around (4,3)
- In OpenGL:

```
glTranslatef( 4, 3 );
glRotatef( 30 );
glTranslatef( -4, -3 );
```



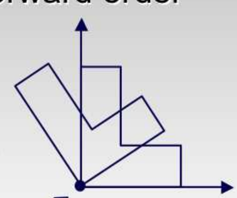


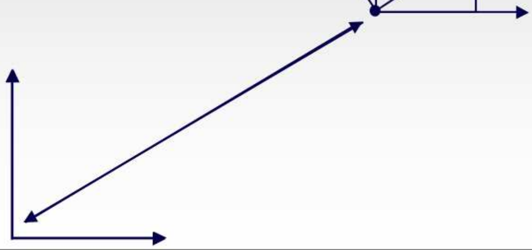
Interpreting Composite Transformations

Interpretation 1: moving the coordinate system

- Read operations in forward order

```
glTranslatef( 4, 3 );
glRotatef( 30 );
glTranslatef( -4, -3 );
```



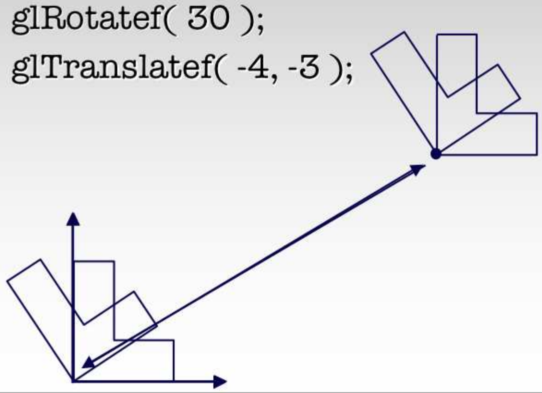


Interpreting Composite Transformations

Interpretation 2: moving the object

- Read operations in reverse order

```
glTranslatef( 4, 3 );
glRotatef( 30 );
glTranslatef( -4, -3 );
```

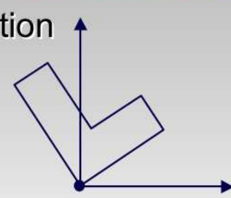


© Wolfgang Heidrich

Compositing of Affine Transformations

Example: Rotation around arbitrary center

- Step 2: perform rotation

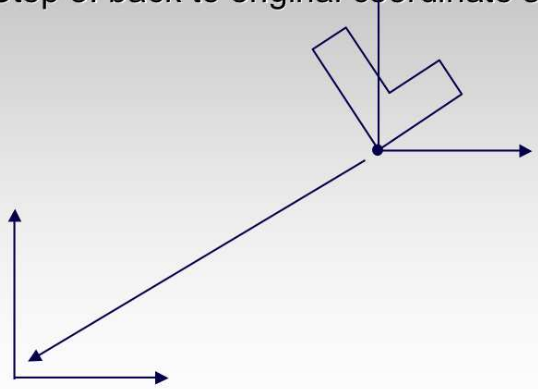


© Wolfgang Heidrich

Compositing of Affine Transformations

Example: Rotation around arbitrary center

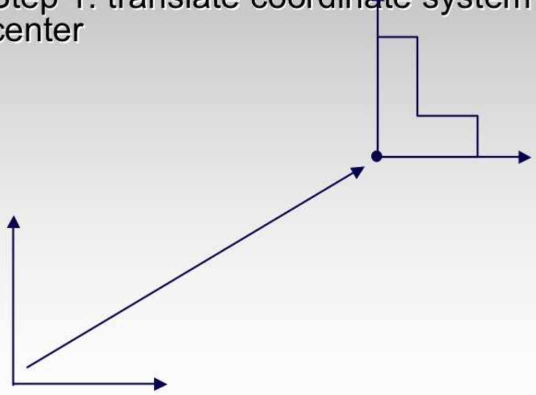
- Step 3: back to original coordinate system



Compositing of Affine Transformations

Example: Rotation around arbitrary center

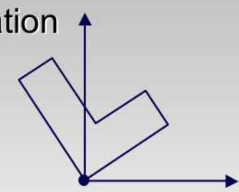
- Step 1: translate coordinate system to rotation center



Compositing of Affine Transformations

Example: Rotation around arbitrary center

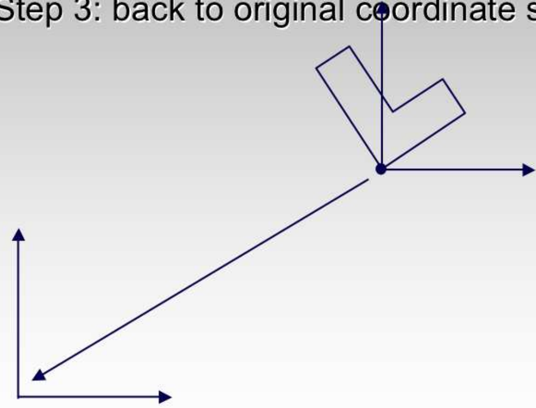
- Step 2: perform rotation



Compositing of Affine Transformations

Example: Rotation around arbitrary center

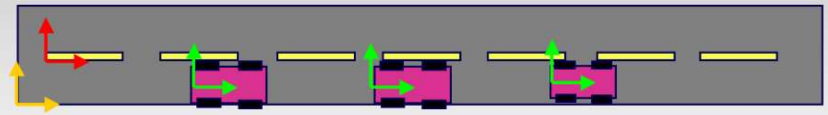
- Step 3: back to original coordinate system



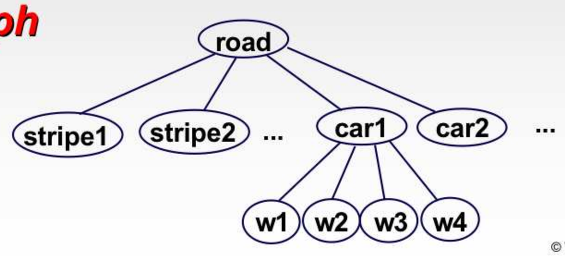
Transformation Hierarchies

Scene may have a hierarchy of coordinate systems

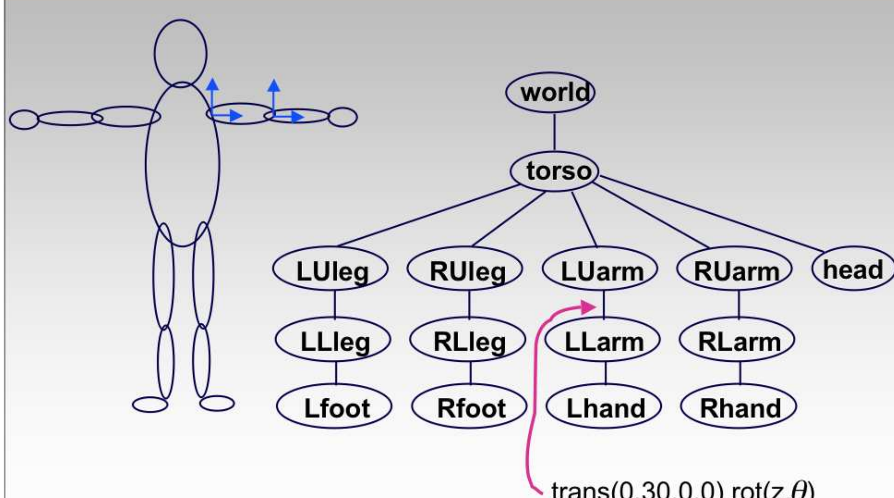
- Stores matrix at each level with incremental transform from parent's coordinate system



Scene graph

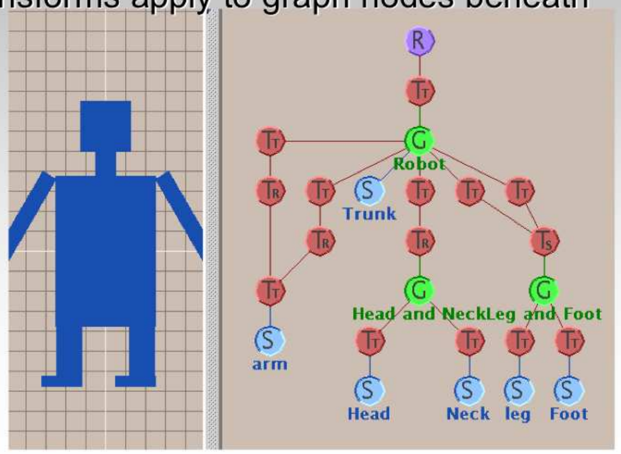


Transformation Hierarchy Example 1



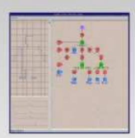
Transformation Hierarchies

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



Brown Applets

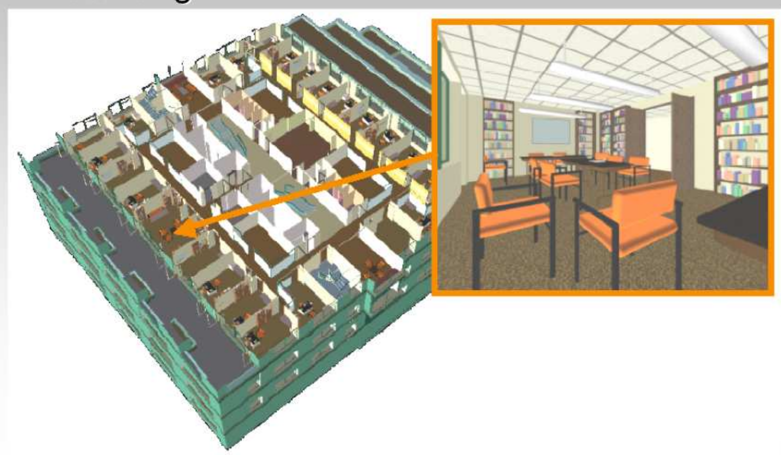
`http://www.cs.brown.edu/exploratories/freeSoftware/catalogs/scenegraphs.html`



- Have a look later

Transformation Hierarchy Example 2

- Draw same 3D data with different transformations: instancing

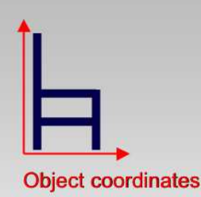


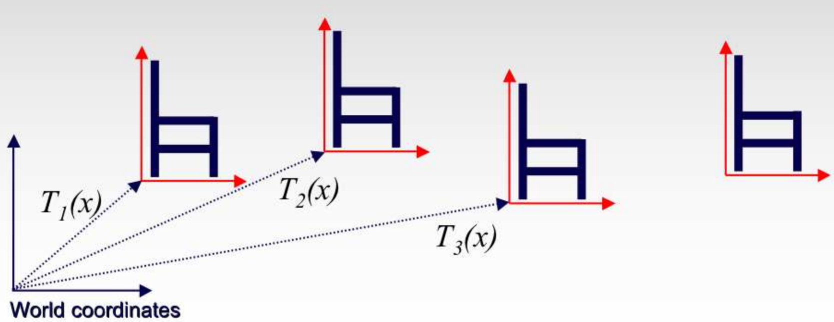
© Wolfgang Heidrich

Matrix Stacks

Challenge of avoiding unnecessary computation

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



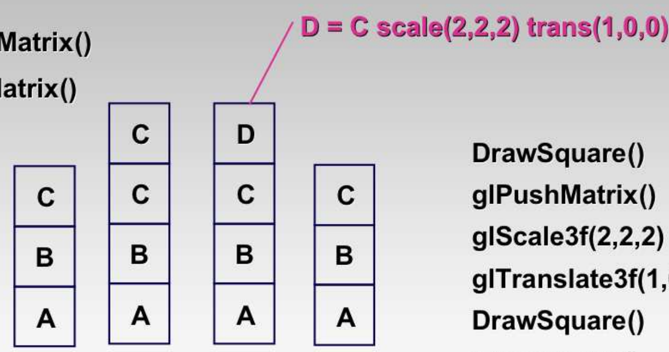


© Wolfgang Heidrich

Matrix Stacks

glPushMatrix()

glPopMatrix()



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

Modularization

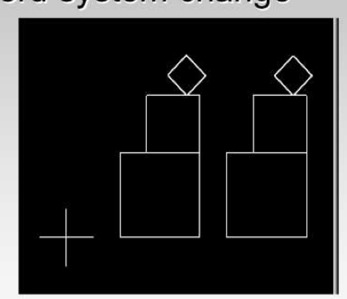
Drawing a scaled square

- 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

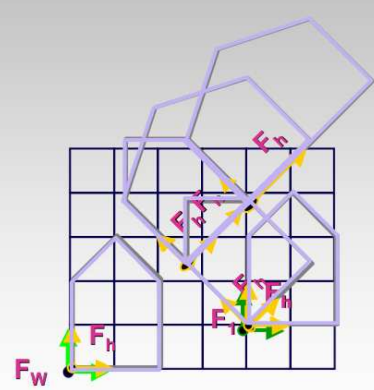
Advantages

- 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

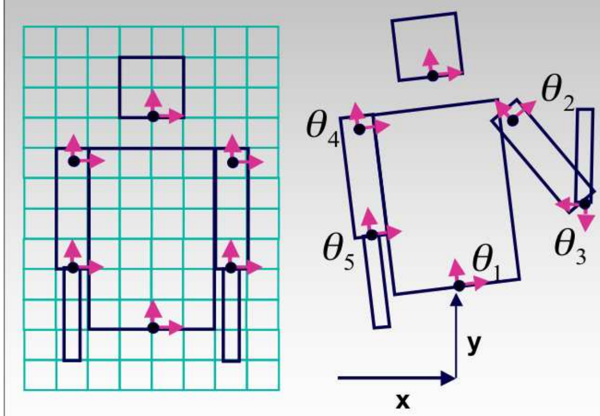
- In graphics hardware, depth of matrix stacks is limited
  - *(typically 16 for model/view and about 4 for projective matrix)*

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(θ1,0,0,1);
DrawBody();
glPushMatrix();
  glTranslate3f(0,7,0);
  DrawHead();
glPopMatrix();
glPushMatrix();
  glTranslate(2.5,5.5,0);
  glRotatef(θ2,0,0,1);
  DrawUArm();
  glTranslate(0,-3.5,0);
  glRotatef(θ3,0,0,1);
  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

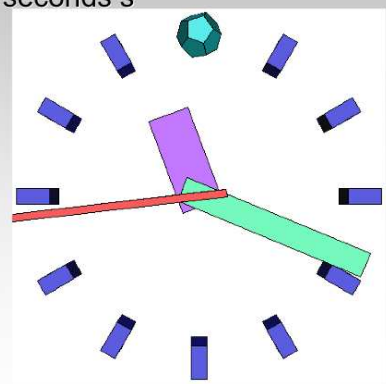
- Expressivity: not always the best controls
- Can't do closed kinematic chains
  - *Keep hand on hip*

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



© Wolfgang Heidrich

Single Parameter: complex

Mechanisms not easily expressible with affine transforms

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

© Wolfgang Heidrich

Display Lists

Concept:

- If multiple copies of an object are required, it can be compiled into a display list:

```
glNewList( listId, GL_COMPILE );
    glBegin( …);
    … // geometry goes here
glEndList();
// render two copies of geometry offset by 1 in z-direction:
glCallList( listId );
glTranslatef( 0.0, 0.0, 1.0 );
glCallList( listId) ;
```


Display Lists

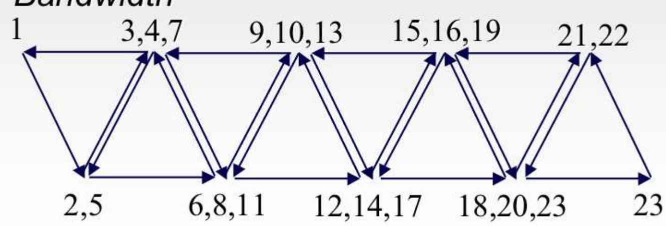
Advantages:

- More efficient than individual function calls for every vertex/attribute
- Can be cached on the graphics board (bandwidth!)
- Display lists exist across multiple frames
  - *Represent static objects in an interactive application*

Shared Vertices

Triangle Meshes

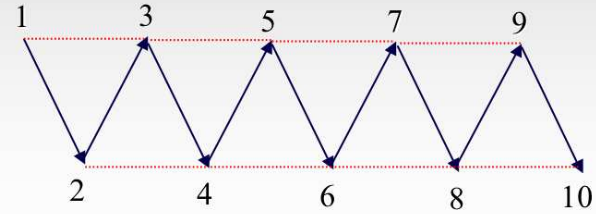
- Multiple triangles share vertices
- If individual triangles are sent to graphics board, every vertex is sent and transformed multiple times!
  - *Computational expense*
  - *Bandwidth*



Triangle Strips

Idea:

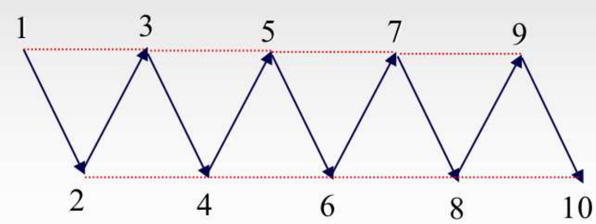
- Encode neighboring triangles that share vertices
- Use an encoding that requires only a constant-sized part of the whole geometry to determine a single triangle
- N triangles need n+2 vertices



Triangle Strips

Orientation:

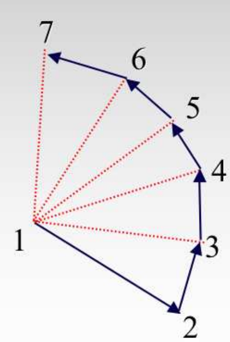
- Strip starts with a counter-clockwise triangle
- Then alternates between clockwise and counter-clockwise



Triangle Fans

Similar concept:

- All triangles share on center vertex
- All other vertices are specified in CCW order



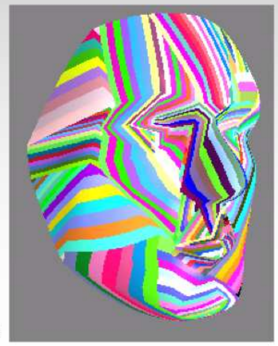
Triangle Strips and Fans

Transformations:

- n+2 for n triangles
- Only requires 3 vertices to be stored according to simple access scheme
- Ideal for pipeline (local knowledge)

Generation

- E.g. from directed edge data structure
- Optimize for longest strips/fans



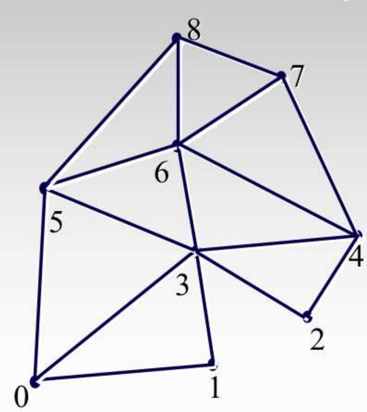
Strippification by Dana Sharon

Vertex Arrays

Concept:

- Store array of vertex data for meshes with aribtrary connectivity (topology)

```
GLfloat *points[3*nvertices];
GLfloat *colors[3*nvertices];
Glint *tris[numtris]=
    {0,1,3,  3,2,4, …};
glVertexPointer( …, points );
glColorPointer( …,colors );
glDrawElements(
    GL_TRIANGLES,…,tris );
```



Vertex Arrays

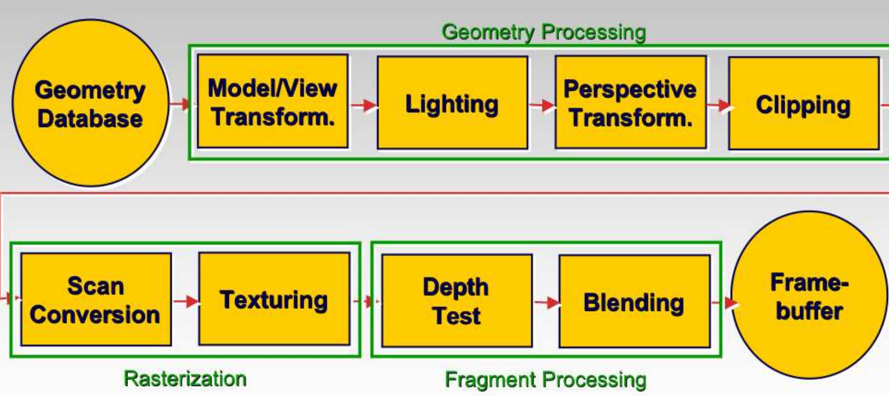
Benefits:

- Ideally, vertex array fits into memory on graphics chip
- Then all vertices are transformed exactly once

In practice:

- Graphics memory may not be sufficient to hold model
- Then either:
  - *Cache only parts of the vertex array on board (may lead to cache trashing!)*
  - *Transform everything in software and just send results for individual triangles (bandwidth problem: multiple transfers of same vertex!)*

The Rendering Pipeline



Coming Up…

Thursday, Sep 20:

- Perspective transformations

Tuesday, Sep 25:

- Lighting