University of British Columbia

Hierarchical modelling

- ***recap: composing transforms***
- ***finish: rendering pipeline***
- ***matrix hierarchies and stacks***
- ***display lists***

News

University of British Columbia

Project 0:

- depth test
  - *glEnable(GL_DEPTH_TEST); // before loop*
  - *glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // in display*
- avoid blob effect: try changing color for each face
- glutDodecahedron **not** the intended solution
  - *good for checking your approach*

Composing transforms recap

University of British Columbia

Order matters

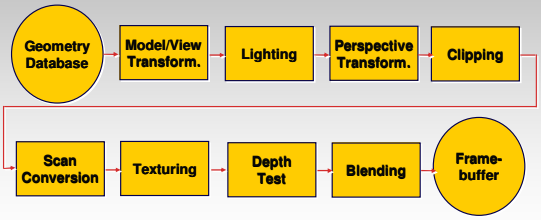
- 4x4 matrix multiplication not commutative!

Moving to origin

- Transformation of geometry into coordinate system where operation becomes simpler
- Perform operation
- Transform geometry back to original coordinate system

The Rendering Pipeline

University of British Columbia



Discussion

University of British Columbia

Advantages of a pipeline structure

- Logical separation of the different components, modularity
- Easy to parallelize:
  - *Earlier stages can already work on new data while later stages still work with previous data*
  - *Similar to pipelining in modern CPUs*
  - *But much more aggressive parallelization possible (special purpose hardware!)*
  - *Important for hardware implementations!*
- Only local knowledge of the scene is necessary

Discussion

University of British Columbia

Disadvantages:

- Limited flexibility
- Some algorithms would require different ordering of pipeline stages
  - *Hard to achieve while still preserving compatibility*
- Only local knowledge of scene is available
  - *Shadows*
  - *Global illumination*

University of British Columbia

Matrix hierarchies and stacks

Matrix Operations in OpenGL

University of British Columbia

Direct matrix specification:

- load
  - *glLoadMatrix*
  - *glLoadIdentity*
- multiply
  - *glMultMatrix*

Transformations:

- glRotate*
- glTranslate*
- glScale*

Matrix Operations in OpenGL

University of British Columbia

glRotate* (θ, x, y, z)

θ **: ccw rotation angle, looking along vector towards origin**

x, y, z **: vector along which rotation occurs**

equivalent to glMultMatrix(A), $A =$

$$\begin{bmatrix} x^2(1-\cos\theta)+\cos\theta & xy(1-\cos\theta)-z\sin\theta & xz(1-\cos\theta)+y\sin\theta & 0 \\ yx(1-\cos\theta)+z\sin\theta & y^2(1-\cos\theta)+\cos\theta & yz(1-\cos\theta)-x\sin\theta & 0 \\ xz(1-\cos\theta)-y\sin\theta & yz(1-\cos\theta)+x\sin\theta & z^2(1-\cos\theta)+\cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matrix Operations in OpenGL

University of British Columbia

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

University of British Columbia

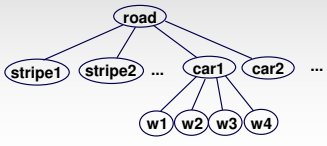
Semantics:

- matrixMode sets the matrix that is to be affected by all following transformations
- Whenever primitives are rendered with glBegin(), the vertices are transformed with whatever the current model/view and perspective matrix is
  - *OpenGL as state machine*
- Multiplication order:
  - *post: right to left, top to bottom*

Transformation Hierarchies

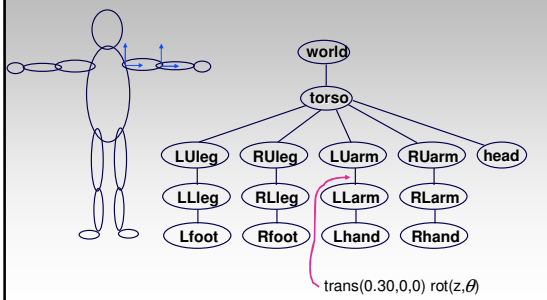
University of British Columbia

scene graph



Transformation Hierarchies

University of British Columbia

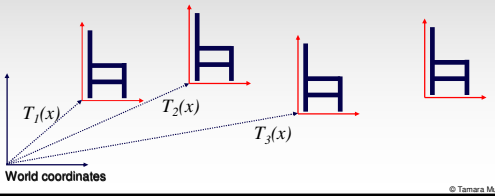


Matrix Stacks

University of British Columbia

Challenge:

- Avoid extra computation
  - *using inverse to return to origin*
  - *computing incremental T_1 -> T_2*



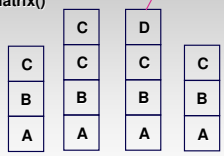
Transformation Hierarchies

University of British Columbia

Matrix Stack

glPushMatrix()
glPopMatrix()

D = C scale(2,2,2) trans(1,0,0)

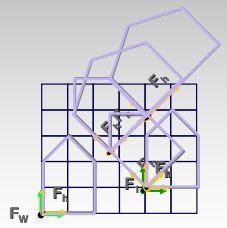



```
DrawSquare()
glPushMatrix()
glScale3f(2,2,2)
glTranslate3f(1,0,0)
DrawSquare()
glPopMatrix()
```


Composing Transformations

University of British Columbia

OpenGL example

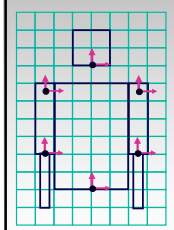


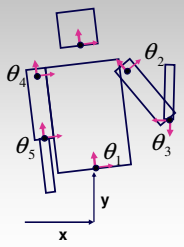

```
glLoadIdentity();
glTranslatef(4,1,0);
glPushMatrix();
glRotatef(45,0,0,1);
glTranslatef(0,2,0);
glScalef(2,1,1);
glTranslate(1,0,0);
glPopMatrix();
```


Transformation Hierarchies

University of British Columbia

Example






```
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: Matrix Stacks

University of British Columbia

Advantages:

- Matrix stacks make it feasible to have multiple copies of one object
- No need to compute inverse matrices all the time
- 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)*

Example: modularization

Drawing a scaled square

```
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();
}
```


Example: applets

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

Display lists

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*

Display Lists

Example: 36 Snowmen

http://www.lighthouse3d.com/opengl/displaylists/

efficiency issues

Snowmen: no lists


```
// Draw 36 Snowmen
for(int i = -3; i < 3; i++)
  for(int j=-3; j < 3; j++) {
  glPushMatrix();
  glTranslatef(i*10.0,0,j * 10.0);
  // Call the function to draw a snowman
  drawSnowMan();
  glPopMatrix();
}
```


Snowmen: no lists


```
void drawSnowMan() {

glColor3f(1.0f, 1.0f, 1.0f);

// Draw Body
glTranslatef(0.0f ,0.75f, 0.0f);
glutSolidSphere(0.75f,20,20);

// Draw Head
glTranslatef(0.0f, 1.0f, 0.0f);
glutSolidSphere(0.25f,20,20);

// Draw Eyes
glPushMatrix();
glColor3f(0.0f,0.0f,0.0f);
glTranslatef(0.05f, 0.10f, 0.18f);
glutSolidSphere(0.05f,10,10);
glTranslatef(-0.1f, 0.0f, 0.0f);
glutSolidSphere(0.05f,10,10);
glPopMatrix();

// Draw Nose
glColor3f(1.0f, 0.5f , 0.5f);
glRotatef(0.0f,1.0f, 0.0f, 0.0f);
glutSolidCone(0.08f,0.5f,10,2);
}
```


Snowmen: no lists


```
// Draw 36 Snowmen
for(int i = -3; i < 3; i++)
  for(int j=-3; j < 3; j++) {
    glPushMatrix();
    glTranslatef(i*10.0,0,j * 10.0);
    // Call the function to draw a snowman
    drawSnowMan();
    glPopMatrix();
  }
```


Display Lists

Example: 36 Snowmen

http://www.lighthouse3d.com/opengl/displaylists/

benchmarks of 36K polygons

- 55 FPS: no display list

Snowmen: display lists


```
GLuint createDL() {
GLuint snowManDL;
// Create the id for the list
snowManDL = glGenLists(1);
// start list
glNewList(snowManDL,GL_COMPILE);
// call the function that contains the rendering commands
drawSnowMan();
// endList
glEndList();
return(snowManDL); }
```


Snowmen: no lists


```
// Draw 36 Snowmen
for(int i = -3; i < 3; i++)
  for(int j=-3; j < 3; j++) {
  glPushMatrix();
  glTranslatef(i*10.0,0,j * 10.0);
  // Call the function to draw a snowman
  glCallList(Dlid);
  glPopMatrix();
}
```

Display Lists

Example: 36 Snowmen

http://www.lighthouse3d.com/opengl/displaylists/

benchmarks of 36K polygons

- 55 FPS: no display list
- 153 FPS: 1 snowman display list, called 36 times

Snowmen: one big list


```
GLuint createDL() {
  GLuint snowManDL;
  snowManDL = glGenLists(1);
  glNewList(snowManDL,GL_COMPILE);
  for(int i = -3; i < 3; i++)
    for(int j=-3; j < 3; j++) {
      glPushMatrix();
      glTranslatef(i*10.0,0,j * 10.0);
      drawSnowMan();
      glPopMatrix();
    }
  glEndList();
  return(snowManDL);
}
```


Display Lists

Example: 36 Snowmen

http://www.lighthouse3d.com/opengl/displaylists/

benchmarks of 36K polygons

- 55 FPS: no display list
- 153 FPS: 1 snowman display list, called 36 times
- 108 FPS: single 36 snowman display list

Snowmen: nested lists


```
GLuint createDL() {
  GLuint snowManDL,loopDL;
  snowManDL = glGenLists(1);
  loopDL = glGenLists(1);
  glNewList(snowManDL,GL_COMPILE);
  drawSnowMan();
  glEndList();
  glNewList(loopDL,GL_COMPILE);
  for(int i = -3; i < 3; i++)
    for(int j=-3; j < 3; j++) {
      glPushMatrix();
      glTranslatef(i*10.0,0,j * 10.0);
      glCallList(snowManDL);
      glPopMatrix();
    }
  glEndList();
  return(loopDL); }
```


Display Lists

Example: 36 Snowmen

http://www.lighthouse3d.com/opengl/displaylists/

benchmarks of 36K polygons

- 55 FPS: no display list
- 153 FPS: 1 snowman display list, called 36 times
- 108 FPS: single 36 snowman display list
- 153 FPS: nested display lists