The University of New Mexico

Introduction to Computer Graphics with WebGL

Ed Angel

Professor Emeritus of Computer Science

Founding Director, Arts, Research, Technology and Science Laboratory

University of New Mexico

WebGL Transformations

Ed Angel
Professor Emeritus of Computer Science
University of New Mexico

Objectives

- Learn how to carry out transformations in WebGL
  - Rotation
  - Translation
  - Scaling
- Introduce MV.js transformations
  - Model-view
  - Projection

The University of New Mexico

Pre 3.1 OpenGL Matrices

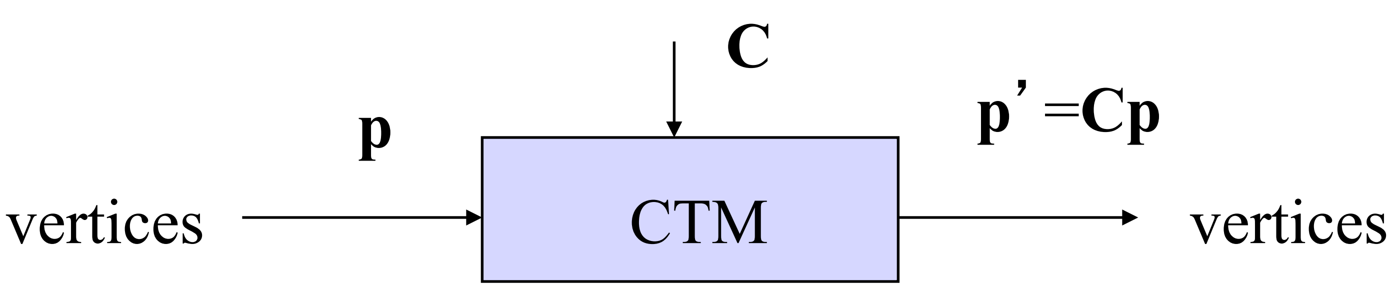
- In Pre 3.1 OpenGL matrices were part of the state
- Multiple types
  - Model-View (**GL_MODELVIEW**)
  - Projection (**GL_PROJECTION**)
  - Texture (**GL_TEXTURE**)
  - Color(**GL_COLOR**)
- Single set of functions for manipulation
- Select which to manipulated by
  - **glMatrixMode(GL_MODELVIEW);**
  - **glMatrixMode(GL_PROJECTION);**

Why Deprecation

- Functions were based on carrying out the operations on the CPU as part of the fixed function pipeline
- Current model-view and projection matrices were automatically applied to all vertices using CPU
- We will use the notion of a **current transformation matrix** with the understanding that it may be applied in the shaders

Current Transformation Matrix (CTM)

- Conceptually there is a 4 x 4 homogeneous coordinate matrix, the *current transformation matrix* (CTM) that is part of the state and is applied to all vertices that pass down the pipeline
- The CTM is defined in the user program and loaded into a transformation unit



CTM operations

- The CTM can be altered either by loading a new CTM or by postmutiplication

Load an identity matrix: $\mathbf{C} \leftarrow \mathbf{I}$
Load an arbitrary matrix: $\mathbf{C} \leftarrow \mathbf{M}$

Load a translation matrix: $\mathbf{C} \leftarrow \mathbf{T}$
Load a rotation matrix: $\mathbf{C} \leftarrow \mathbf{R}$
Load a scaling matrix: $\mathbf{C} \leftarrow \mathbf{S}$

Postmultiply by an arbitrary matrix: $\mathbf{C} \leftarrow \mathbf{CM}$
Postmultiply by a translation matrix: $\mathbf{C} \leftarrow \mathbf{CT}$
Postmultiply by a rotation matrix: $\mathbf{C} \leftarrow \mathbf{C\,R}$
Postmultiply by a scaling matrix: $\mathbf{C} \leftarrow \mathbf{C\,S}$

The University of New Mexico

Rotation about a Fixed Point

Start with identity matrix: $\mathbf{C} \leftarrow \mathbf{I}$
Move fixed point to origin: $\mathbf{C} \leftarrow \mathbf{CT}$
Rotate: $\mathbf{C} \leftarrow \mathbf{CR}$
Move fixed point back: $\mathbf{C} \leftarrow \mathbf{CT}^{-1}$

Result: $\mathbf{C} = \mathbf{TR}\,\mathbf{T}^{-1}$ which is **backwards**.

This result is a consequence of doing postmultiplications.
Let's try again.

Reversing the Order

We want $\mathbf{C} = \mathbf{T}^{-1}\, \mathbf{R}\, \mathbf{T}$
so we must do the operations in the following order

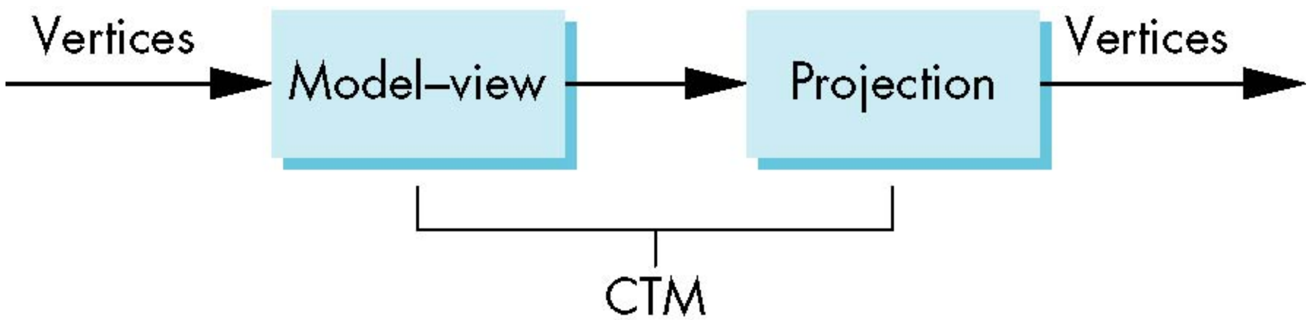
$\mathbf{C} \leftarrow \mathbf{I}$
$\mathbf{C} \leftarrow \mathbf{C}\mathbf{T}^{-1}$
$\mathbf{C} \leftarrow \mathbf{C}\mathbf{R}$
$\mathbf{C} \leftarrow \mathbf{C}\mathbf{T}$

Each operation corresponds to one function call in the program.

Note that the last operation specified is the first executed in the program

CTM in WebGL

- OpenGL had a model-view and a projection matrix in the pipeline which were concatenated together to form the CTM
- We will emulate this process



Using the ModelView Matrix

- In WebGL, the model-view matrix is used to
  - Position the camera
    - Can be done by rotations and translations but is often easier to use the lookAt function in MV.js
  - Build models of objects
- The projection matrix is used to define the view volume and to select a camera lens
- Although these matrices are no longer part of the OpenGL state, it is usually a good strategy to create them in our own applications

$$q = P*MV*p$$

Rotation, Translation, Scaling

Create an identity matrix:

```
var m = mat4();
```

Multiply on right by rotation matrix of **theta** in degrees where (**vx, vy, vz**) define axis of rotation

```
var r = rotate(theta, vx, vy, vz)
m = mult(m, r);
```

Also have rotateX, rotateY, rotateZ
Do same with translation and scaling:

```
var s = scale( sx, sy, sz)
var t = translate(dx, dy, dz);
m = mult(s, t);
```

Example

- Rotation about z axis by 30 degrees with a fixed point of (1.0, 2.0, 3.0)

```
var m = mult(translate(1.0, 2.0, 3.0),
    rotate(30.0, 0.0, 0.0, 1.0));
m = mult(m, translate(-1.0, -2.0, -3.0));
```

- Remember that last matrix specified in the program is the first applied

Arbitrary Matrices

- Can load and multiply by matrices defined in the application program
- Matrices are stored as one dimensional array of 16 elements by MV.js but can be treated as 4 x 4 matrices in row major order
- OpenGL wants column major data
- gl.unifromMatrix4f has a parameter for automatic transpose by it must be set to false.
- flatten function converts to column major order which is required by WebGL functions

Matrix Stacks

- In many situations we want to save transformation matrices for use later
  - Traversing hierarchical data structures (Chapter 9)
- Pre 3.1 OpenGL maintained stacks for each type of matrix
- Easy to create the same functionality in JS
  - push and pop are part of Array object

```
var stack = [ ]
stack.push(modelViewMatrix);
modelViewMatrix = stack.pop();
```