Supplement to Tutorial
JavaScript Library for Matrix and Vector Operations

If you have not finished the programs of the previous weeks, work on them first. We are moving to drawing in 3D, and it is essential that you understand the structures and control follow of the WebGL programs we have practiced so far.

1. Introduction

The JavaScript itself does not support matrix operations. To perform matrix operations in WebGL, we need to use a JavaScript library developed by a third party. The commonly used libraries in WebGL applications are Sylvester, WebGL-mjs and glMatrix. These libraries are similar. We use glMatrix (`glMatrix.js`).

The glMatrix was written primarily for WebGL. It supports three element vector, 3x3 and 4x4 matrix and quaternion operations. See the reference at the end of this handout for the functions in the library.

The convention of notation of glMatrix matrix is different from the normal mathematical convention. For example, the translation matrix in normal matrix form

$$\begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

is written in glMatrix as

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x & y & z & 1 \end{bmatrix}$$

But this is only a notational difference.

2. Use the library functions

To create a vector with three elements, you use

```
vec3.create(vec);
```

where `vec` is an optional array containing the three elements you want to initialize the vector with. For example, the following creates a three-component vector with the elements 1, 2, and 3:

```
var u = vec3.create([1,2,3]);
```

You can also create a three-component vector without initializing it:

```
var u = vec3.create();
```

Most of the functions in glMatrix that perform operations on vectors or matrices have an optional last argument that is the destination vector or destination matrix for the operation. Functions that operate on a vector have the following form:

vec3.someOperation(*srcVec, otherOperands, destVec*(*optional*))*;*

If *destVec* is specified, the result of the operation is written to *destVec*, which is returned from the function. However, if *destVec* is not specified, the result of the operation is instead written to *srcVec* and also returned.

In the following code snippet, you can see how it works when the optional destination argument is specified. Two vectors `u` and `v` are created and initialized with values. Then a third vector s is created to hold the result of the add operation. Since the third vector will only be used to write the result to, you do not need to initialize it. Finally, the two vectors are added and the result is written to the destination vector `s`:

```
var u = vec3.create([1,2,3]);
var v = vec3.create([4,5,6]);
var s = vec3.create();
vec3.add(u,v,s); // s = [5,7,9] and u is unchanged
```

Since the optional destination vector is specified, the result is written to this vector, and this is the only argument that is changed.

If the destination vector is not specified, the result of addition will be written back to the first argument:

```
var u = vec3.create([1,2,3]);
var v = vec3.create([4,5,6]);
var s = vec3.add(u,v); // s = [5,7,9]
                       //and u = [5,7,9]
```

Operations on matrices basically follow the same pattern. You can create a 4 × 4 matrix with a call to the function,

```
    mat4.create(mat);
```

where `mat` is an optional array that contains the 16 elements that you want to initialize the matrix with. The ***first four elements*** in the array populate the ***first column*** of the matrix, the next four elements populate the second column, and so on.

If you want to specify the optional initialization array, the code would look like this:

```
var N = mat4.create([0,1,2,3,    // first column
                     4,5,6,7,    // second column
                     8,9,0,1,    // third column
                     2,3,4,5]); // fourth column
```

If you have two matrices M and N and want to calculate the product **MN**, you would call

```
var P = mat4.multiply(M,N);
```

Note that since the optional destination matrix is not specified for this multiplication, the result would be written to matrix `M`, and to P.

The program at the end of this document shows the source code of an example program that uses the glMatrix library. In addition to the usual vector and matrix manipulation, the example also uses the function `vec3.str()` and `mat4.str()` to convert a vector or a matrix to a string. Pay attention to how function `mat4.translate()` is used. It creates a translation matrix that translates two units in x-direction, three units in y-direction, and four units in z-direction. The elements that correspond to the translation have the indices 13, 14, and 15. JavaScript *alert* boxes are used for the output.

Exercise: Matrix Operations.
Copy `glMatrix.js` into your working directory.

```
<!DOCTYPE HTML>
<html lang="en">
<head>
<title>Test of glMatrix JavaScript Library</title>
<script type="text/javascript" src="glMatrix.js"></script>

<script type="text/javascript">

function testglMatrixJsLibrary() {
  var u = vec3.create([1,2,3]);
  var v = vec3.create([4,5,6]);
  var r = vec3.create();
  var t = vec3.create([1,2,3]);

/* In the add below the optional receiver vec3 is specified as 3:rd
argument so the first two arguments are not modified */
  var s = vec3.add(u,v,r);  // s = r = [5,7,9]
  alert(vec3.str(s));       // will alert [5,7,9]
  alert(vec3.str(r));       // will alert [5,7,9]
  alert(vec3.str(u));       // will alert [1,2,3]

  // In the add below the optional receiver vec3 is not specified
  // so the the result is written to the first argument u in
  // addition to s2
  var s2 = vec3.add(u,v)  // s2 = [5,7,9] as expected,
                          // but now also u = [5,7,9]
  alert(vec3.str(s2));    // will alert [5,7,9]
  alert(vec3.str(u));     // will alert [5,7,9]

  var d = vec3.dot(t,v);  // d = 1*4+2*5+3*6 = 32
  alert(d);               // will alert 32

  var c = vec3.cross(t,v,r); // c = r = [-3,6,-3]
  alert(vec3.str(r));        // will alert [-3,6,-3]

  var I = mat4.create([1,0,0,0,   // first column
                       0,1,0,0,   // second column
                       0,0,1,0,   // third column
                       0,0,0,1]); // fourth column

  var M = mat4.create([1,0,0,0,   // first column
                       0,1,0,0,   // second column
                       0,0,1,0,   // third column
                       2,3,4,1]); // fourth column

  var IM = mat4.create();
  mat4.multiply(I, M, IM);
  alert(mat4.str(IM));  // will alert [1,0,0,0
                        //             0,1,0,0,
                        //             0,0,1,0,
                        //             2,3,4,1]
  var T = mat4.create();
  mat4.translate(I, [2,3,4],T);
  alert(mat4.str(T));   // will alert [1,0,0,0
                        //            0,1,0,0,
                        //            0,0,1,0,
                        //            2,3,4,1]
}

</script>
</head>
<body onload="testglMatrixJsLibrary();">
  Simple web page to test glMatrix JavaScript library. </br>
  The page shows a couple of alerts with the result
  from vector and matrix maths. </br>
  You need to read the source code to understand the results of the
alerts.
</body>
</html>
```

3. Quick reference of the functions

Three dimensional vector operations

vec3.create(vec) Creates a new instance of a vec3 from **vec** that contains 3 numeric values.
vec3.set(vec, dest) Copies **vec** to **dest**.

vec3.add(vec, vec2, dest): adds **vec** and **vec2** and write to **dest**.
vec3.subtract(vec, vec2, dest) subtracts **vec** by **vec2.**
vec3.negate(vec, dest) Negates the components of **vec.**
vec3.scale(vec, val, dest) uniformly scale vec by value val.
vec3.normalize(vec, dest) normalise vex (make it of unit length).
vec3.cross(vec, vec2, dest) calculate the cross product of vex and vec2.
vec3.length(vec) Caclulates the length of a vec3.
vec3.dot(vec, vec2) Caclulates the dot product of vec and vec2.
vec3.lerp(vec, vec2, lerp, dest) Performs a linear interpolation between two vex and vec2. lerp specifics the amount of interpolation (between 0.0 and 1.0).
vec3.str(vec) Returns a string representation of a vector, vec.

3x3 Matrix (mat3) operations

mat3.create(mat) Creates a new instance of a 3x3 matrix from mat, which containing at 9 numeric values.
mat3.set(mat, dest) Copies the values of mat to dest.
mat3.identity(dest) Create an identity matrix.
mat3.transpose(mat, dest) Transposes mat.
mat3.toMat4(mat, dest) Copies the elements of a mat3, mat, into the upper 3x3 elements of a mat4, dest.
mat3.str(mat) Returns a string representation of a mat3.

4x4 Matrix (mat4) operations

mat4.create(mat)
mat4.set(mat, dest)
mat4.str(mat) Returns a string representation of a mat4.
mat4.identity(dest)
mat4.transpose(mat, dest)
mat4.determinant(mat) Calculates the determinant of a mat4.
mat4.inverse(mat, dest) Calculates the inverse matrix of a mat4.
mat4.toRotationMat(mat, dest) Copies the upper 3x3 elements of a mat4, mat into another mat4, dest
mat4.toMat3(mat, dest) Copies the upper 3x3 elements of a mat4 into a mat3.
mat4.toInverseMat3(mat, dest) Calculates the inverse of the upper 3x3 elements of a mat4 and copies the result into a mat3. The resulting matrix is useful for calculating transformed normals.
mat4.multiply(mat, mat2, dest)
mat4.multiplyVec3(mat, vec, dest) Transforms a vec3 with the given matrix. The 4th vector component is implicitly '1'.
mat4.multiplyVec4(mat, vec, dest) Transforms a vec4 with the given matrix.
mat4.translate(mat, vec, dest) Translates a matrix by the given vector.
mat4.scale(mat, vec, dest) Scales a matrix by the given vector. vec - vec3 specifying the scale for each axis.
mat4.rotate(mat, angle, axis, dest) Rotates a matrix by the given angle around the specified axis. mat - mat4 to rotate, angle - angle (in radians) to rotate, and axis - vec3 representing the axis to rotate around.
mat4.rotateX(mat, angle, dest) Rotates a matrix by the given angle around the X axis. angle - angle (in radians) to rotate.
mat4.rotateY(mat, angle, dest)
mat4.rotateZ(mat, angle, dest)
mat4.frustum(left, right, bottom, top, near, far, dest) Generates a frustum matrix with the given bounds: left, right - scalar, left and right bounds of the frustum. bottom, top - scalar, bottom and top bounds of the frustum. near, far - scalar, near and far bounds of the frustum. dest - Optional, mat4 frustum matrix will be written into.
mat4.perspective(fovy, aspect, near, far, dest) Generates a perspective projection matrix with the given bounds. fovy - scalar, vertical field of view in degrees. aspect - scalar, aspect ratio =viewport width/height.near, far - scalar, near and far bounds of the frustum. dest - Optional, mat4 frustum matrix will be written into.

mat4.ortho(left, right, bottom, top, near, far, dest) Generates a orthogonal projection matrix with the given bounds. left, right - scalar, left and right bounds of the frustum. bottom, top - scalar, bottom and top bounds of the frustum. near, far - scalar, near and far bounds of the frustum. dest - Optional, mat4 frustum matrix will be written into.

mat4.lookAt(eye, center, up, dest) Generates a look-at matrix with the given eye position, focal point, and up axis. eye - vec3, position of the viewer. center - vec3, point the viewer is looking at. up - vec3 pointing "up". dest - Optional, mat4 frustum matrix will be written into.

Quaternion (quat4) operations

quat4.create(quat) Creates a new instance of a quat4 using quat that contains at least 4 numeric elements can serve as a quat4.

quat4.set(quat, dest) Copies the values of one quat4 to another.

quat4.str(quat) Returns a string representation of a quaternion.

quat4.calculateW(quat, dest) Calculates the W component of a quat4 from the X, Y, and Z components. Assumes that quaternion is 1 unit in length. Any existing W component will be ignored.

quat4.inverse(quat, dest) Calculates the inverse of a quat4.

quat4.length(quat) Calculates the length of a quat4.

quat4.normalize(quat, dest) Generates a unit quaternion of the same direction as the provided quat4.

quat4.multiply(quat, quat2, dest) Performs a quaternion multiplication.

quat4.multiplyVec3(quat, vec, dest) Transforms a vec3 with the given quaternion.

quat4.toMat3(quat, dest) Calculates a 3x3 matrix from the given quat4.

quat4.toMat4(quat, dest) Calculates a 4x4 matrix from the given quat4.

quat4.slerp(quat, quat2, slerp, dest) Performs a spherical linear interpolation between two quat4.