## **8. Move and rotate CG using GLUT**

Perform CG drawing with polygons in OpenGL.

Unfortunately, this lesson does not explain how to determine colors from light and reflection, so we will specify RGB values ​​directly for colors.

(I used to teach it, but since it was too much, I excluded the learning of the technique of rendering, which is the procedure of setting the color from the light property and the reflectance of the surface of the object)

**Reference link**

·[OpenGL functions (functions that begin with the two-letter gl lowercase)](http://www.opengl.org/sdk/docs/man/)

·[OpenGLProgramming Guide Book (ver1.1, commonly known as red](http://glprogramming.com/red/index.html) book)

[·GLUT - The OpenGL Utility Toolkit](http://www.opengl.org/resources/libraries/glut/)

[·](http://www.opengl.org/resources/libraries/glut/)[Toolkit (GLUT) Programming Interface API Version 3](http://www.opengl.org/resources/libraries/glut/spec3/spec3.html)

### **08.01. Matrices in OpenGL Library**

OpenGL uses GL\_PROJECTION matrix and GL\_MODELVIEW matrix on CG rendering.

Now, here, the GL\_PROJECTION matrix is ​​denoted by **P.**  The GL\_MODELVIEW matrix is denoted by **M.**

The position vector of one point in a certain 3D space (where I want to use CG) can usually be represented by a 3D coordinate vector.

(Students who remembered linear algebra should have thought, "What? You miss some important information.")

.(Students who remember all the contents of section 06.05. students can say "Ah, I found them in the 06.05.")

(In linear algebra classes, the "three-dimensional space" where we live is a “linear space”. A "position" in the space is called a "vector”.)

(If you don't remember these definitions, you’d better reviewing the "position vector" in Chapter 1 of the linear algebra textbook. You have to remember them.)

(So, the problem is that we need something like a major premise to express a "vector" in a "coordinate vector" form, right?)

(......)

(Yes, we have to prepare a “base”. A coordinate representation without showing its base does not make sense at all.)

(OpenGL uses a “orthonormal base”. The explanation is written in 06.05.)

(A linear space where a "orthonormal base" can be set is a “metric linear space”, also known as an “inner product space”.)

(So, to describe the orthonormal base, we need to know where is the origin. What is the focus of the camera that becomes the origin? How are the three base vectors facing where they orthogonal to each other? How is "one unit" defined? Such questions should be posed one after another.)

(By the way, this difference in the definition of the orientation of the base is one of the beginnings of the unwanted battle between the OpenGL and DirectX groups. I wonder why they do not share the common definition. For details, let's find what I am telling you by yourself by starting to search it with the keywords of "right-handed system" and "left-handed system")

As .explained in Section 06.05., OpenGL expresses the coordinate vector as a 4-dimensional coordinate vector.

(By doing this, the drawing algorithm of 3D CG can be described overwhelmingly efficiently.)

(This was the major invention for realizing realtime CG. This and matrix accumulation. This linear algebraic knowledge made real-time graphics possible.)

**GL\_PROJECTION matrix**

Reference: [Projection Transformation](http://glprogramming.com/red/chapter03.html#name3)

Consider the projection of a position vector b in the camera three dimensional coordinate system (see 06.05.) onto the position vector a in the image plane (a.k.a. anthoer linear space with less dimensions).

The position vector b in the camera three dimensional coordinate system can be express by the coordinate vector *b* by refering the orthonormal basis in the space.

This coordinate system representation is extended to represent b as a four dimensional coordinate vector.

For now, the four-dimensional coordinate vector should be presented as (X, Y, Z, 1)T. The fourth element is always set to 1.

The projected position vector a exists in a two-dimensional linear space called the image plane.

Using a orthonormal basis set in the image coordinate system, a position a is represented by the two-dimensional coordinate vector *a*.

This coordinate system representation is also extended to represent a as a four-dimensional coordinate vector.

At the moment, the coordinate of the point a after the projection should be *a* = (u, v, f, 1)T. The fourth element should be 1.

Although f is the value corresponding to the focal length of the camera, please think of it as a constant in this class.

(From here, when we mention a position vector, it is shown in the form of the coordinate vector.)

Normally, CG imaging is a projection from linear 3D to linear 2D, so it is a linear mapping.

The coordinate system has been expanded so that the coordinate vector of the projection source and the coordinate vector of the projection destination are aligned in four dimensions.

In other words, this map is apparently a linear transformation. The representation matrix becomes a square matrix.

This representation matrix is ​​called the GL\_PROJECTION matrix in OpenGL.

Here, a projection matrix corresponding to the orthographic projection in Section 06.05 is denoted by **P**ortho.

*a* = **P**ortho *b*

**GL\_MODELVIEW Matrix**

Reference: [Viewing and Modeling Transformations](http://glprogramming.com/red/chapter03.html#name2)

The role of the GL\_MODELVIEW matrix is ​​to change the placement of the CG object with respect to the camera by moving or rotating the object.

(Now I'm still using the expression "move the model in front of the camera")

GL\_MODELVIEW matrix is denoted as **M.**

Let the position vector c before moving and the position vector *b* after the moving.

*b* = **M** *c*

When nothing is moved or rotated, **M** is an identity matrix.

**Matrix operations**

These two operations can be summarized as follows.

Consider a position *c* on the 3D model in the 3D model coordinate system (expanded to four-dimensional coordinate vector).

The position *c* is being moved and rotated by **M**. to a position *b*. Then *b* is projected to *a* by **P**.

*a* = **P**ortho *b* = **P**ortho (**M** *c*) = **P**ortho **M** *c*

**Checking the matrix**

Let's programmatically check what the values ​​of the GL\_PROJECTION matrix and the GL\_MODELVIEW matrix actually.

OpenGL uses a float one-dimensional array (16 elements) to represent a 4x4 matrix. (08-01-ex3)

Actually, the GL\_PROJECTION matrix and the GL\_MODELVIEW matrix cannot be referenced directly in the OpenGL library.

(I won't write the reason here. That is beyond the scope of this class.)

Instead, use glGetFloatv() to assign it to a variable you have prepared.

(It's a similar idea to the sscanf() function)

Use glGetFloatv() to create a function that displays the GL\_PROJECTION matrix and the GL\_MODELVIEW matrix together.

It's not close to any function groups so far, so I'll have a separate source file for this.

What state of the matrix should we know by using this function?

It's just before you start drawing the object as CG. This will be explained in the next section.

[Program 08-01] (Gray part is unchanged))

1. ic2-CommonHeaders.h (Difference from 07-0508-01
2. -Rendering.c (Difference from)
3. 07-0508-01-GLTools.c (New )
4. 07-03-EmbededObjects.c, 07-03-Projection.c, 07-04-Initialization.c, 07-05-Callback.c, 07-05-MainFunction.c

**About project settings in eclipse**

On Section 07 and after, especially after this section 08, the number of files that do not need to be rewritten will increase.

(If the files that do not need to be rewritten can be completely fixed, we will make a library, but I will not do that in this class because I will change the source files here and there.)

Please prepare a new project for each section as follows.

1. First, create a new project (for example, “08-01” here).

2. Place the required new source files in that project directory (~/workspace-ipd/08-01/).

3. Source files inherited from the previous projects without rewriting (in this case, the source files in section 07.05., which are not newly rewritten) (~/workspace-ipd/07-05/\*.[Ch])should be copied to the project folder (~/workspace-ipd/08-01/). Do this outside of eclipse.

(There is nothing you can't do in eclipse, but ...)

4. "Update" the project.

5. Don't forget to set the library to link.

6. If you are concerned that the (inherited) source file name is still old, right-click on the file name you want to change in the project tree, select "Rename", and give it an appropriate name.

(Eclipse automatically changes the compilation method etc. when the file name changes)

**Exercise**

**08-01-ex1**: Prediction of projection matrix

Predict the value of orthographic projection matrix of Section 06.05. **P**ortho before you actually check the value by the program.

**08-01-ex2**: GL\_MODELVIEW matrix

Show the value of the GL\_MODELVIEW matrix as in its initial state and explain the reason.

**08-01-ex3**: Arrangement of 4x4 matrix in OpenGL

Explain how the 16 elements of float m [16]; are arranged to express corresponding 4x4 matrix in OpenGL. Pay particular attention to the subscripts.

**08-01-ex4**: Description of glGetFloatv() function

Find the document of the description of glGetFloatv() function in "[Explanation of each OpenGL function](http://www.opengl.org/sdk/docs/man/)" and show the URL and its explanation part. (Simply showing http://www.opengl.org/sdk/docs/man/ is insufficient.)

**08-01-ex5**: New OpenGL library functions

List all the newly introduced OpenGL library functions in this section. Explain how to use them one by one briefly.

**08-01-ex6**: Logo replacement to your original logo

Replace the contents of ic2\_OpenGLLogo() with your original logo created by 06-06-ex3.

**08-01-ex7**: Printf() in ic2\_ShowMATRIX()

Add ffush() after all the printf()s in ic2\_ShowMATRIX().

### **08.02. First move**

Reference: [Viewing and Model in Transformations](http://glprogramming.com/red/chapter03.html#name2)

The only CG object we still have is the object drawn by ic2\_OpenGLLogo().

Let's move this object 0.5 to the right. (08-02-ex1)

First, let's move by +0.01 along the X axis of the camera coordinate system.

(You may wonder why 0.01 is while saying 0.5, but there is a good reason to come out later.)

**Concept of transformation**

This transformation will be realized by manipulating the GL\_MODELVIEW matrix **M**move..

*a* = **P**ortho **M**move *c*

Use glTranslatef() for the function that represents the transformation. (08-02-ex2)

The three arguments of glTranslatef(X, Y, Z) indicate the amount of transformation.

Perhaps, when we first think about the transformation of an object, we want to realize it by "moving the object in the camera coordinate system".

However, in this OpenGL programming, the coordinate values ​​in the ic2\_OpenGLLogo() function are not changed.

(If you want to "move", you should prepare the coordinates of each vertex added by 0.5 in the X direction, but it is very troublesome to write the program source like that. The coordinates are in the ic2\_OpenGLLogo() function. Rewriting seems to be so many.)

In the ic2\_OpenGLLogo() function, the coordinate values are set ​​according to the "model coordinate system".

(The horizontal axis and vertical axis of the square paper used when designing the logo mark correspond to this "model coordinate system".)

(The horizontal one unit and the vertical one unit corresponds to the orthonormal base in this linear space. )

The actual mapping can be described in the following:

*a* = **P**ortho *b* = **P**ortho (**M**move) *c*) = **P**ortho **M**move *c*

**Matrix representing transformation**

Executing the glTranslatef(X, Y, Z) produces the 4x4 matrix where X, Y, Z are arranged in the 4th column of the row 1, 2, and 3 in addition to the identity matrix.

This is the **M**move.

**Implementation of transformation**

According to the above explanation, glTranslatef(X, Y, Z) can be used anywhere before the actual drawing starts with ic2\_OpenGLLogo().

Here, let's operate just before calling ic2\_OpenGLLogo().

In OpenGL, when using a function that manipulates matrices such as glTranslatef(), you must declare which of the four OpenGL matrices to manipulate.

To do this, run glMatrixMode(GL\_MODELVIEW) just before glTranslatef() to indicate that subsequent operations are for GL\_MODELVIEW matrices.

[Program 08-02] (Gray part is unchanged)

1. 08-02-Rendering.c (Difference from 08-01)
2. 07-03-EmbededObjects.c, 07-03-Projection.c, 07-04-Initialization.c, 07-05-Callback.c, 07-05-MainFunction.c, 08-01-GLTools.c, ic2-CommandHeaders.h

**Execute**

Pay attention to the GL\_MODELVIEW matrix that is displayed.... Whoa, what’s happening?!

**Exercise**

**08-02-ex1**: Units

What is the unit of "0.01" in the above explanation?

**08-02-ex2**: glTranslatef() function description

Find the description of glTranslatef() function in "[Explanation of each OpenGL function](http://www.opengl.org/sdk/docs/man/)" and indicate the URL. If you cannot specify the URL, you can also use the URL + procedure.

"https://www.opengl.org/sdk/docs/man2/" is not sufficient for the answer.

**08-02-ex3**: Test of moving amount

Try changing (0.01, 0, 0) of moving amounts in various ways. Pay particular attention to what happens if you change the Z value. Capture the resulting images and explain the status of them in text.

**08-02-ex4**: Confirmation of unexpected transformation

Confirm the moves of the logo in the window as the program goes.

### **08.03. Actual form of matrix operation in OpenGL**

Reference:[Viewing and Modeling Transformations](http://glprogramming.com/red/chapter03.html#name2)

In the previous section, animation was realized unexpectedly.

Why did that happen?

In fact, matrix manipulation in OpenGL has one feature.

That is, the matrix operation is "performed on the previous calculation result so far".

**Matrix accumulation**

When an OpenGL program starts, as seen in 08.01., The GL\_MODELVIEW matrix **M**st is an identity matrix.

**M**st ← **I**

When glTranslatef() is executed for the first time, if the moving matrix defined by the options of glTranslatef() is presented by **T**, **M**st becomes:

**M**st ← **M**st. **T**

Actually, the resultant **M**st is **T** here.

Next, when it comes to the stage where glTranslatef () is executed again, since **M**st is not initialized, the operation comes to:

**M**st ← **M**st  **T**

And , the resultant matrix **M**st becomes **T T**, that is, the amount of transformation is doubled.

For the third time, the transformation comes to triple.

(Not that this operation is not like **M**st ← **T** **M**st but **M**st ← **M**st  **T**. Since the commutative law is not valid on matrix operation, this is important.)

**Eliminating the matrix accumulation**

How can we stop this unwilling animation?

Before the operation, **M**st should be explicitly initialized by the identity matrix.

**M**st ← **I**

**M**st ← **M**st **T**

The glLoadIdentity() function is a special function that “load” the initial matrix. It should be placed just before executing glTranslatef().

**Programming**

Let's check the operation with programming.

For confirmation, try embedding ic2\_showMATRIX() in 3 places.

(You should not say that the execution result of the program is rather boring)

[Program 08-03] (The gray part is unchanged)

1. 08-03-Rendering.c (Difference from 08-02)
2. 07-03- EmbededObjects.c, 07-03-Projection.c, 07-04-Initialization.c, 07-05-Callback.c, 07-05-MainFunction.c, 08-01-GLTools.c, ic2-CommandHeaders.h

**Exercise**

**08-03-ex1**: Change the X delta

Change the X delta value from 0.01 to different value such as 0.03.

**08-03-ex2**: Moving matrix

As explained in Section 08.01., In OpenGL, 3D spatial coordinates are represented by (X, Y, Z, 1)T. For transformation, a transformation matrix is ​​prepared by the glTranslatef() function. At this time, X, Y, Z indicate the amount of transformation.

Explain how this moving matrix is ​​represented on the MODELVIEW matrix.

You can also use the URL found in 08-02-ex2.

Reference <http://glprogramming.com/red/appendixf.html>

**08-03-ex3**: Rotation matrix

A rotation matrix is prepared by the glRotate\*(a, X, Y, Z) function. At this time, a indicates the rotation angle, and X, Y, Z indicate the rotation axis vector. This is so called axis-angle representation.

Find out how this rotation matrix is ​​represented on the MODELVIEW matrix.

Reference <http://glprogramming.com/red/appendixf.html>

**08-03-ex4**: Orthogonal matrix

a Rotation matrix is an orthogonal matrix in linear algebra. Show everything as long as you remember the nature of the orthogonal matrix.

(It has useful and convenient properties on CG programming.)

(For example, what can be guaranteed if all the matrices for coordinate transformation are orthogonal matrices?)

(In other words, what happens if non-orthogonal matrices are included in the series of matrix operations?)

(What was the nature of the orthogonal transformation on which the orthogonal matrix is ​​derived?)

(Knowledge of mathematics is directly linked to the examination of applicability here.)

**08-03-ex5**: Inverse rotation matrix

Now, think of a matrix that rotates by “a” around the axis (X, Y, Z). On the other hand, think of a matrix that rotates around the same axis by “-a”. How should these be written in OpenGL function? And what is the relationship between the two matrices obtained in this way?

**08-03-ex6**: Matrix product arithmetic law

Explain the "associative law", "commutative law", and "distributive law" of general arithmetic laws. Show which of these three rules is unsuccessful in matrix multiplication.

**08-03-ex7**: Exceptions to matrix product arithmetic laws

In general, arithmetic laws that should not be successful in matrix multiplication may be successful under certain conditions. Identify the arithmetic law and explain the specific conditions where the arithmetic law turns to be successful.

(At least one example is actually taken up in this lesson)

**08-03-ex8**: Checking the state of GL\_MODELVIEW matrix

Check how the GL\_MODELVIEW matrix is modified as X delta variable "0.01" is accumulated in the matrix when the program progresses.

(In the system of this lesson, it may be necessary to set the fflush () function as in Section 05.08. To update the display as it is executed on Eclipse.)

### **08.04. Multi-step matrix operation**

OpenGL enables complex operations by expressing transformation operations in multi-step matrix operations.

There are three main operations: translate, rotate, and zoom in / out.

Both of these are represented by a matrix. (08-04-ex1)

Here, let's realize the operation of Figure-3.4 in [Viewing and Modeling Transformations](http://glprogramming.com/red/chapter03.html#name2) section.

Rotate it 45 ° counterclockwise on the Z axis, and then translate it 0.7 along the X axis in the camera coordinate system. (See the left of data/08-04-fig00.png).

If the rotation matrix is **M**rot and the moving matrix is **M**move, then it can be expressed by:

**M**st ← **I** **M**move **M**rot

Note that the matrix sequence is inverted from the description sequence.

(Don't make it in reverse order!)

This is because the position vector to be manipulated is placed to the right of this and applied from the matrix on the right.

In the OpenGL implementation, as mentioned in the previous section, the matrix that is calculated later is calculated from the right with respect to the matrix that precedes it.

This means that the matrix manipulation functions are arranged in reverse order with respect to the descriptive word order (from left to right in the formula).

Assuming that R represents the rotation matrix and T represents the transformation matrix, before programming, consider whether RT or TR is correct.

(Once again, note that the position vector is multiplied from the right side of the matrix.)

matrix operations are generally non-exchangeable (associative and distributive laws hold).

Implementation of [RT]

glRotatef(45, 0, 0, 1);  
glTranslatef(0.7, 0, 0);

Implementation of [TR]

glTranslatef(0.7, 0, 0);  
glRotatef(45, 0, 0, 1);

In the glRotatef() function that represents the rotation matrix, the first argument is the rotation angle (specified by degree), and the second to fourth arguments indicate the direction vector of the rotation axis. (08-03-ex3)

[Program 08-04] (Gray part is unchanged)

1. 08-04-Rendering.c (Difference from 08-03)
2. 07-03-EmbededObjects.c, 07-03-Projection.c , 07-04-Initialization.c, 07-05-Callback.c, 07-05-MainFunction.c, 08-01-GLTools.c, ic2-CommandHeaders.h

If it's hard to understand, let's use the key 's' / 'S'.

(The change due to the logoscale variable is directly sizing the drawing coordinate value in ic2\_OpenGLLogo(), so this operation is completed within the model coordinate system. In other words, the CG object itself is made smaller. Therefore, the matrix operation has nothing to do with this scaling operation.)

(It may be easier to understand if the scale is set to 0.1, which is smaller than 0.5.)

**Exercise**

**08-04-ex1**: Matrix operation

Implement both RT and TR. Execute them and find the difference.

**08-04-ex2**: glRotatef() · Clockwise

Use only glRotatef() to create a program that slowly rotates the logo clockwise.

**08-04-ex3**: glRotatef () · Y-axis

Use only glRotatef() to create a program where the logo slowly rotates around the Y-axis (with the right side coming front first).

**08-04-ex4**: Uncomfortable feeling of 08-04-ex3

Explain the difference between the intuitive expectation and the appearance of 080-04-ex3 and tell the reason.

**08-04-ex5**: X-axis rotation direction in glRotatef() function

In glRotatef() function, draw a picture to show the rotation direction when a positive rotation value is given to the X-axis vector (1,0,0). Also draw the direction of the camera's line of sight in the picture.

**08-04-ex6**: Y-axis rotation direction in glRotatef() function

In glRotatef () function, draw a picture to show the rotation direction when a positive rotation value is given to the Y-axis vector (0,1,0). Also draw the direction of the camera's line of sight in the picture.

**08-04-ex7**: Z-axis rotation direction in glRotatef() function

In glRotatef() function, draw a picture to show the rotation direction when a positive rotation value is given to the Z-axis vector (0,0,1). Also draw the direction of the camera's line of sight in the picture.

**08-04-ex8**: Rotation of the logo

Create a program in which the small logo moves clockwise at a distance from the center of the window (Motion like the logo is attached to the tip of the second hand on a watch).

**08-04-ex9**: Rotation of the stabilized logo

Create a program in which the small logo moves clockwise at a distance from the center of the window, but do not rotate the logo this time (it is like watching a Ferris wheel from the side).