3D Хувиргалтууд

Lecture 7

Outline

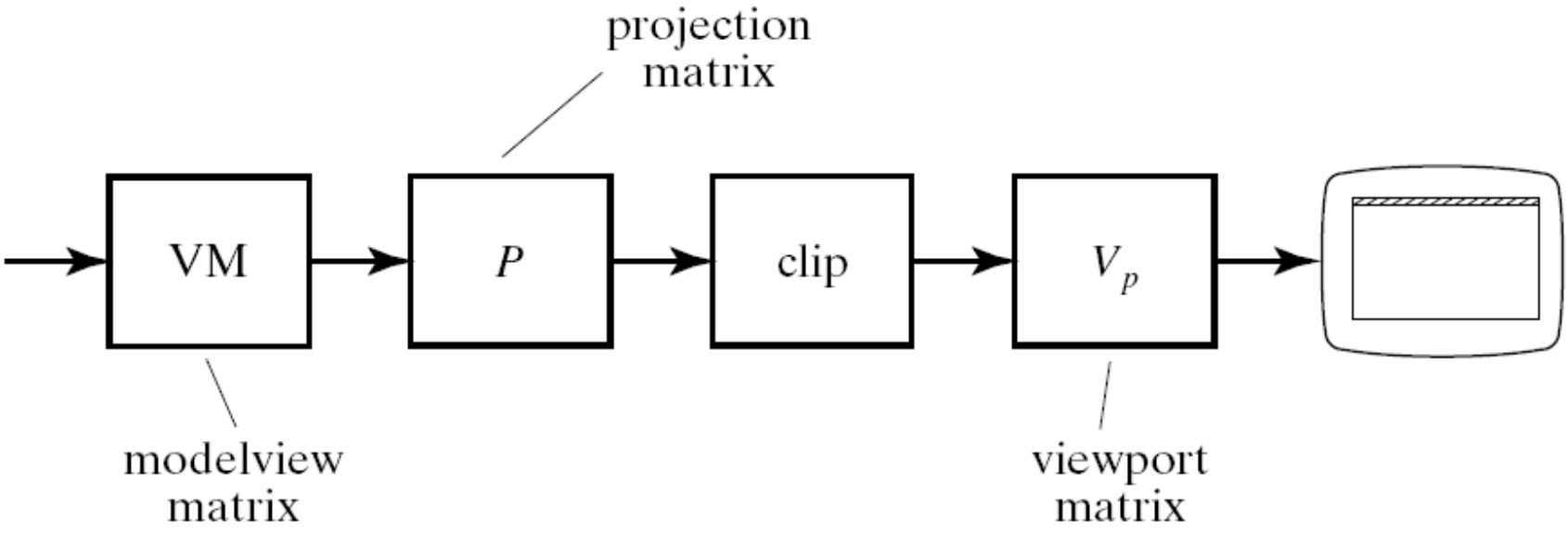
- Model View Matrix
- Translation Matrix
- Rotation Matrix
- Scaling Matrix
- Identity Matrix
- Matrix Stacks

Хэрэгцээ, шаардлага

- Объектүүдийг загварчлах, байрлуулах болон өөр хооронд нь холбож нэгтгэхэд хувиргалтууд/transformations/ хийх шаардлагатай болдог
- Жишээ нь бидэнд 1 данх/гүц байна гэж бодвол
  - Түүнийг бусад юмстай харьцангуйгаар хаана байрлуулах зөв байрлалыг олох
  - Аль өнцгөөс нь харуулах уу, эсвэл харах уу
  - Одоо байгаа хэмжээнээс нь томсгох уу жижигрүүлэх үү гэх мэт шаардлага тулгарна.

The OpenGL Pipeline (simplified)

OpenGL-ийн графикын хоолойг хялбаршуулсан зураглал



Modeling/Загварчлал

- 3D огторгуй дахь биетийн загвар x-, y-, z-координатууд бүхий оройнуудаар тодорхойлогдох бөгөөд цэг /point/, шулуун /line/, олон өнцөгтийн /polygon/ аль нэгээр дүрслэгдэнэ.
- Оройнуудыг зөөх, эргүүлэх, скэйлдэх зориулалт бүхий олон тооны функц OpenGL санд байдаг.
- Эдгээр хувиргалтыг “**affine transformations**” гэнэ
- OpenGL эдгээр аффин хувиргалтуудыг тодорхойлохдоо “**model view matrix**” гэж нэрлэгдэх 4x4 хэмжээтэй матрицүүдийг ашигладаг.

Viewing/Харагдац

- 3 хэмжээст загварын харагдах байдлыг удирдахад гол төлөв ашигладаг.
- Камерыг тавих буюу харах цэг, чиглэлийг заана
- Графикт үндсэн 2 төрлийн буулгалт/**projection**/ хэрэглэдэг.
  - Orthographic
  - Perspective
- Харагдацыг тодорхойлох хувиргалтын хувьд мөн 4x4 хэмжээтэй өөр нэг матрицыг OpenGL-д ашигладаг.

OpenGL дэх хувиргалтын матрицууд

- OpenGL-д glVertex()-ээр тодорхойлогдсон оройнуудын байршлуудыг хувиргах үүрэг бүхий model view матриц-тэй ажилладаг хэд хэдэн функцүүд байдаг.

glMatrixMode(GL_MODELVIEW);

- Selects the **Model View Matrix (M)** as the currently active matrix
- Other matrices that can be selected with this command include the **Projection Matrix**
- The Projection Matrix and the Model View Matrix work together to position object

- Оройнуудыг байршуулахдаа эдгээр 2 матрицыг хамтад нь хэрэглэдэг.

Modeling Transformation Commands

- glLoadIdentity()
  - Sets M equal to the 4 x 4 identity matrix
- glTranslatef (float u_x, float u_y, float u_z)
  - This command sets M equal to $M \cdot T$, where T is the Transformation Matrix

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Modeling Transformation Commands

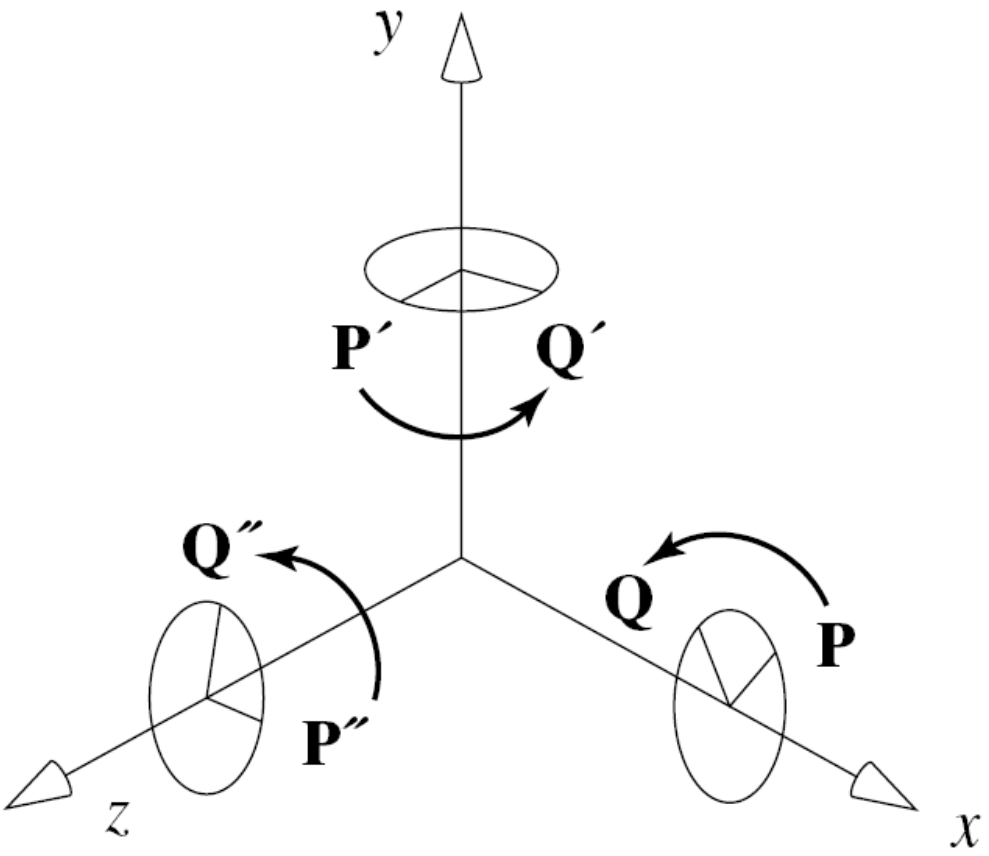
- **glRotatef(float θ, float x, float y, float z)**
  - Sets M equal to M·R(θ, u), where θ is the rotation angle and **u**=(u_x, u_y, u_z) is unit vector

$$\begin{pmatrix} (1-c)u_x^2+c & (1-c)u_xu_y-su_z & (1-c)u_xu_z+su_y & 0 \\ (1-c)u_xu_y+su_z & (1-c)u_y^2+c & (1-c)u_yu_z-su_x & 0 \\ (1-c)u_xu_z-su_y & (1-c)u_yu_z+su_x & (1-c)u_z^2+c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where c=cos θ, s=sin θ

Positive rotations about the three axes

Гурван тэнхлэгийн орчим дахь эерэг эргүүлэлтүүд



Derivation of the Rotation Matrix

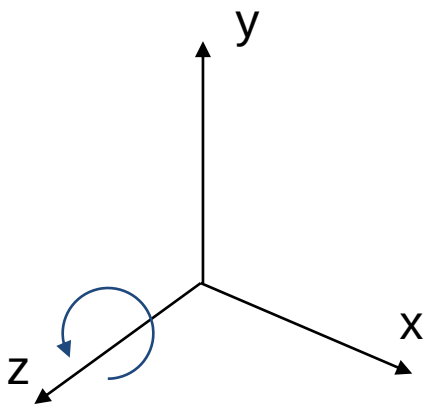
z тэнхлэгийн орчин дахь эргэлт

- Rotation around **z-axis**

  $x' = x \cos\theta - y \sin\theta$

  $y' = x \sin\theta + y \cos\theta$

  $z' = z$



$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

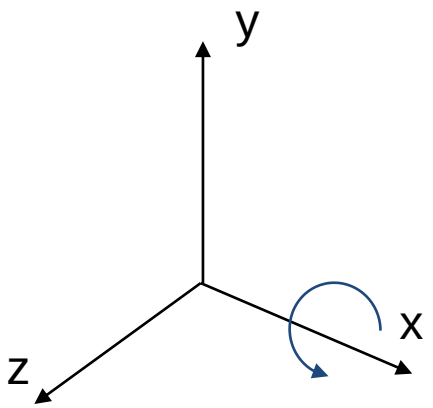
Derivation of the Rotation Matrix
x тэнхлэгийн орчин дахь эргэлт

- Rotation around **x-axis**

  $y' = y \cos\theta - z \sin\theta$

  $z' = y \sin\theta + z \cos\theta$

  $x' = x$



$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

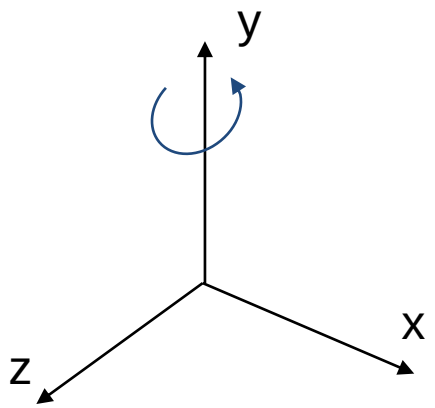
Derivation of the Rotation Matrix
y тэнхлэгийн орчин дахь эргэлт

- Rotation around **y-axis**

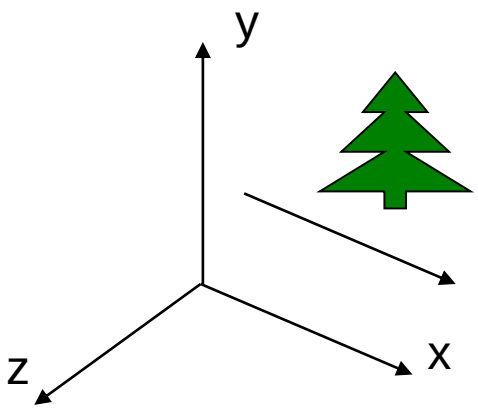
  $z' = z \cos\theta - x \sin\theta$

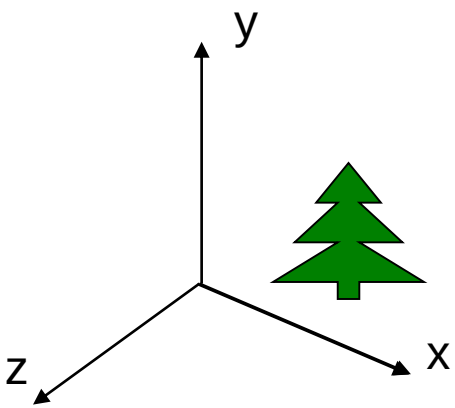
  $x' = z \sin\theta + x \cos\theta$

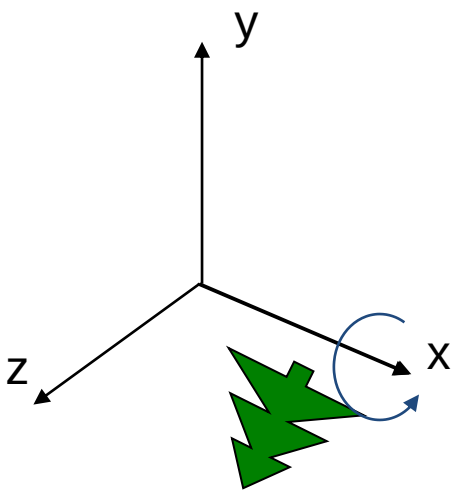
  $y' = y$



$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$



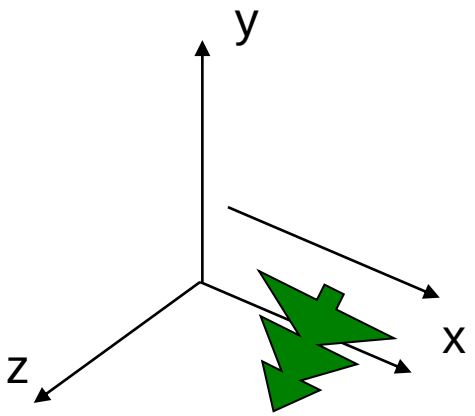


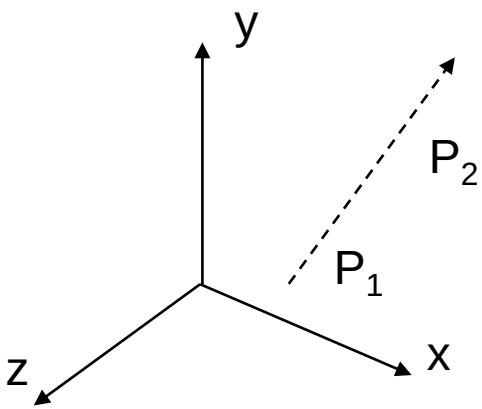


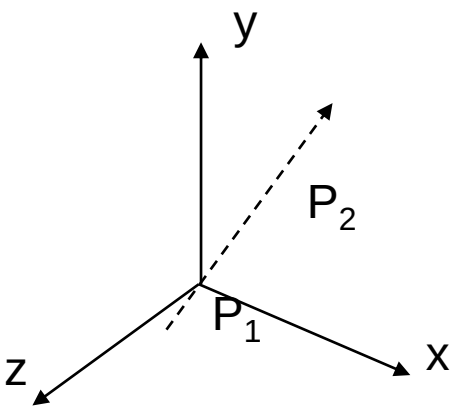
$$P' = T^{-1} \cdot R_x(\theta) \cdot T \cdot P$$

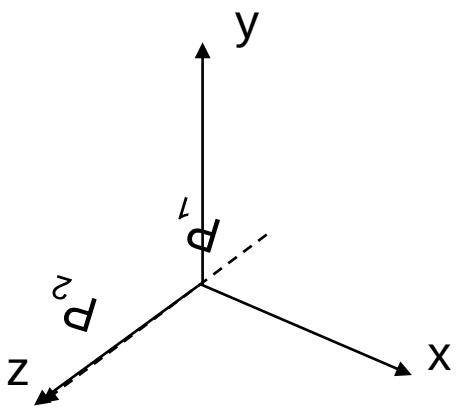
$$R(\theta) = T^{-1} \cdot R_x(\theta) \cdot T$$

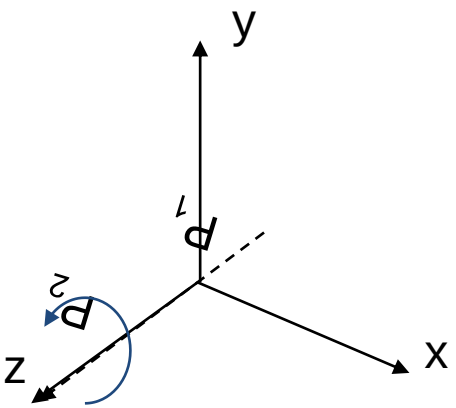
$$P' = R(\theta) \cdot P$$

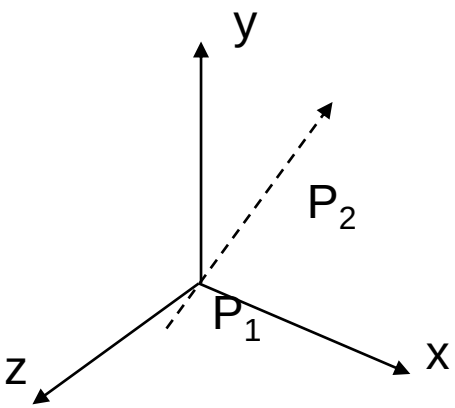


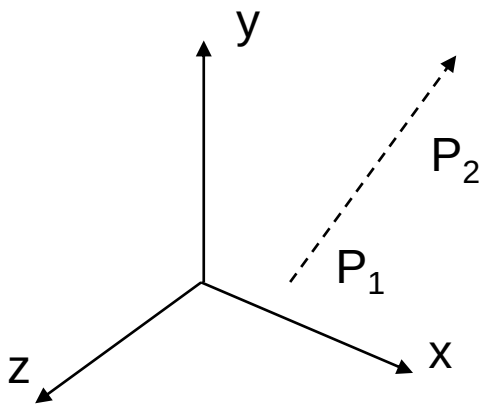

y
P2
P1
z
x


y
P2
P1
z
x


y
P1
P2
z
x


y
P1
P2
z
x


y
P2
P1
z
x


y
P2
P1
z
x

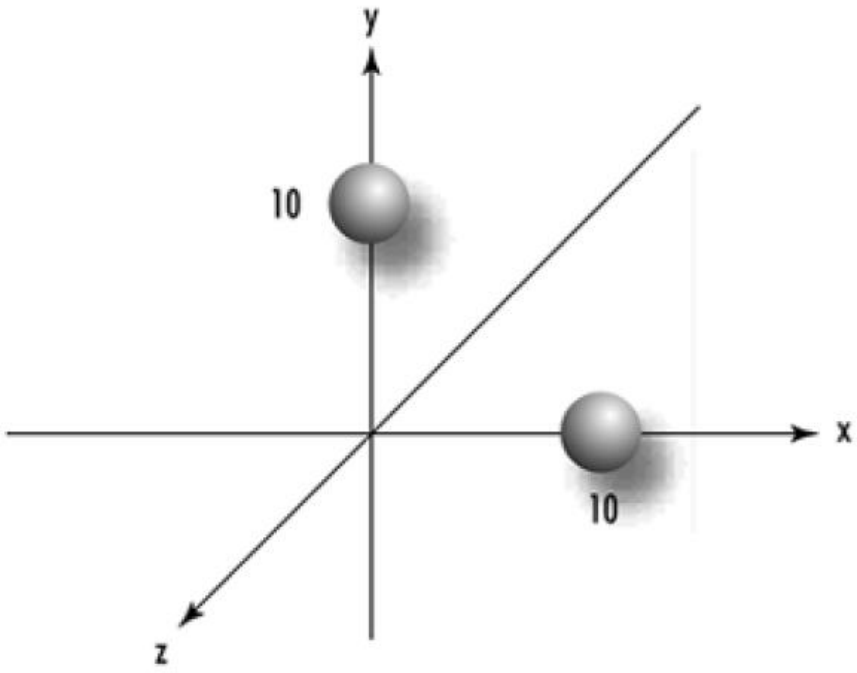
Modeling Transformation Commands

- **glScalef (float s_x, float s_y, float s_z)**
  - This command scales the x-, y-, z-coordinates of points independently.
  - That is to say, it sets M=M·S, where S is the **Scaling Matrix**

$$\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

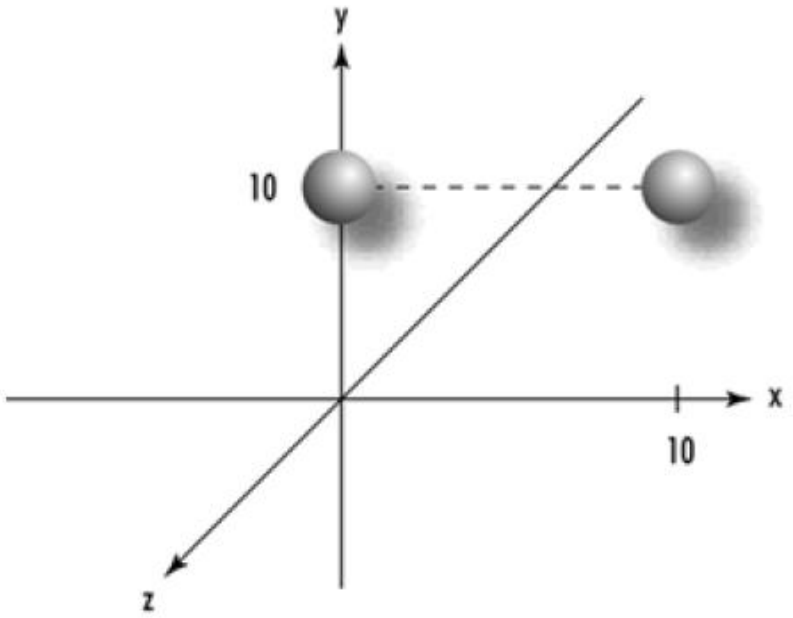
The Identity Matrix
Нэгж матриц

- Suppose we want to draw two spheres as shown in figure.



The Identity Matrix
Нэгж матриц

```
// Go 10 units up the y-axis
glTranslatef(0.0f, 10.0f, 0.0f);
// Draw the first sphere
glutSolidSphere(1.0f,15,15);
// Go 10 units out the x-axis
glTranslatef(10.0f, 0.0f, 0.0f);
// Draw the second sphere
glutSolidSphere(1.0f);
```



The Identity Matrix
Нэгж матриц

- You should reset the origin by loading the modelview matrix with the identity matrix.
- The identity matrix specifies that no transformation is to occur, in effect saying that all the coordinates you specify when drawing are in eye coordinates.

The Identity Matrix
Нэгж матриц

- The following two lines load the identity matrix into the modelview matrix:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

  - The first line specifies that the current operating matrix is the modelview matrix.
  - The second line loads the current matrix with the identity matrix.

Now, the following code produces correct result

```
// Set current matrix to modelview and reset
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

// Go 10 units up the y-axis
glTranslatef(0.0f, 10.0f, 0.0f);
// Draw the first sphere
glutSolidSphere(1.0f, 15, 15);

// Reset modelview matrix again
glLoadIdentity();

// Go 10 units out the x-axis
glTranslatef(10.0f, 0.0f, 0.0f);
// Draw the second sphere
glutSolidSphere(1.0f, 15, 15);
```

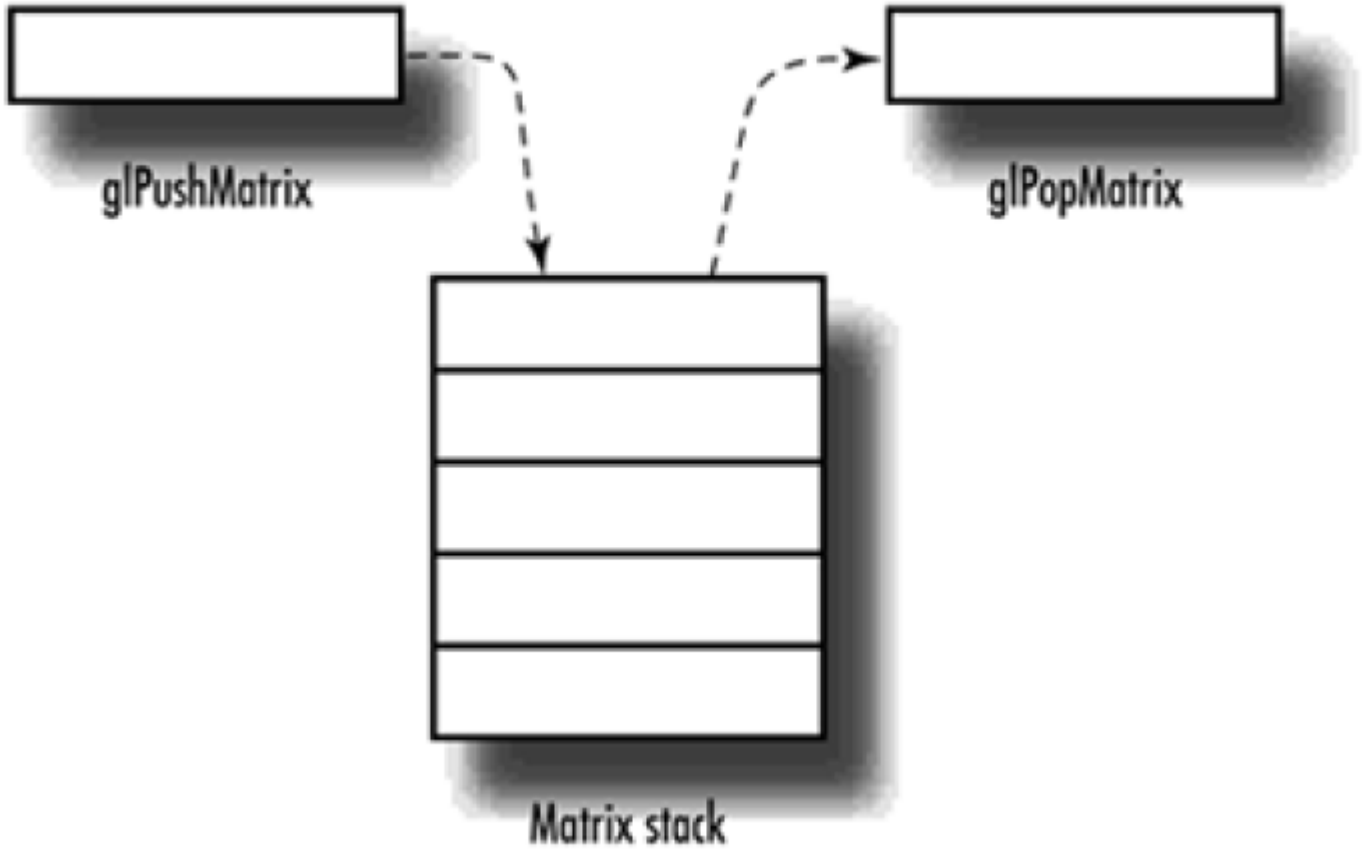
The Matrix Stacks
Матрицын стек

- Resetting the modelview matrix to identity *before placing every object* is not always desirable.
- Often, you want to save the current transformation state and then restore it after some objects have been placed. This approach is most convenient when you have initially transformed the modelview matrix as your viewing transformation

The Matrix Stacks

- To facilitate this procedure, OpenGL maintains a *matrix stack* for both the modelview and projection matrices.
- A matrix stack works just like an ordinary program stack.
- You can push the current matrix onto the stack to save it and then make your changes to the current matrix.
- Popping the matrix off the stack restores it.

The Matrix Stacks
Матрицын стек



Rewrite the code using the Matrix Stacks

```
// Set current matrix to modelview and reset
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

glPushMatrix();
    glTranslatef(0.0f, 10.0f, 0.0f);
    glutSolidSphere(1.0f, 15, 15);
glPopMatrix();

glPushMatrix();
    glTranslatef(10.0f, 0.0f, 0.0f);
    glutSolidSphere(1.0f, 15, 15);
glPopMatrix();
```

- OpenGL includes the following two commands that allow you to use any homogenous **4 x 4** matrix you wish
- Both of these commands take *16 floating point numbers as inputs* and create a **4 x 4** homogeneous matrix with components
- The elements of the matrix are given in **column order**!
  - **glLoadMatrixf (float * matEntries);**
    - This initializes M to be the matrix with entries the 16 numbers pointed to by *matEntries*
  - **glMultMatrixf(float * matEntries);**
    - This sets M equal to M·M‘, where M‘ is the matrix with entries equal to the 16 values pointed to by *matEntries*

matEntries

- The variable *matEntries* can have its type defined by any one of the followings:
  - float *matEntries;
  - float matEntries[16];
  - float matEntries[4][4];

    - In the third case, the entry in row ***i*** and column ***j*** is the value **matEntries[*j*][*i*]**

glLoadMatrixf()

```
// Load an identity matrix
glFloat m[] = {
    1.0f, 0.0f, 0.0f, 0.0f, // X Column
    0.0f, 1.0f, 0.0f, 0.0f, // Y Column
    0.0f, 0.0f, 1.0f, 0.0f, // Z Column
    0.0f, 0.0f, 0.0f, 1.0f // Translation
};
glMatrixMode(GL_MODELVIEW);
glLoadMatrixf(m);
```