Computer Graphics
- Primitive Transformations

Sung Soo Hwang

Primitive Transformations

- Transformations
  - To create and move the composite model using Geometric primitives (cube, sphere, cone, cylinder, torus), you will learn how to implement three Transformations in OpenGL code.

  - Composite Transformation Matrix:
    - $[T] = [T_n] \cdot \cdots [T_3] \cdot [T_2] \cdot [T_1]$

←

Multiplication Order matters

Review Transformation Functions

- Rotation
  - Required Header file: **<glm/gtc/matrix_transform.hpp>**

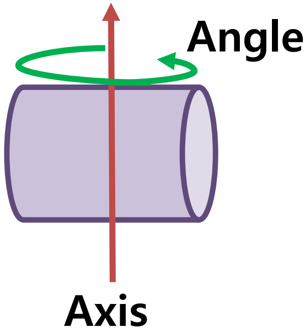
```
mat4 rotate(mat4 const & m, float angle, vec3 const & axis);
dmat4 rotate(dmat4 const & m, double angle, dvec3 const & axis);
```

Existing transf. mat.

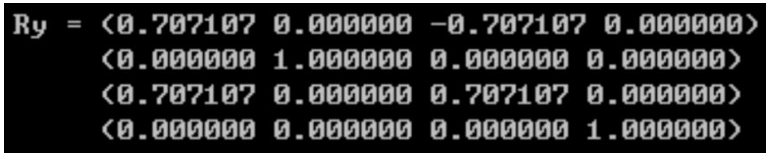
$$[T] = m \cdot [T_r]$$

Transf. mat. for rotation w.r.t. **axis** by **angle**

```
mat4 Ry(1.0f);
Ry = rotate(Ry, radians(45.0f), vec3(0.0f, 1.0f, 0.0f));
printf(“ Ry = (%f %f %f %f)\n”, Ry[0][0], Ry[0][1], Ry[0][2], Ry[0][3]);
printf(“        (%f %f %f %f)\n”, Ry[1][0], Ry[1][1], Ry[1][2], Ry[1][3]);
printf(“        (%f %f %f %f)\n”, Ry[2][0], Ry[2][1], Ry[2][2], Ry[2][3]);
printf(“        (%f %f %f %f)\n”, Ry[3][0], Ry[3][1], Ry[3][2], Ry[3][3]);
```



$$R_y(45°) = \begin{bmatrix} \cos(45°) & 0 & \sin(45°) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(45°) & 0 & \cos(45°) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

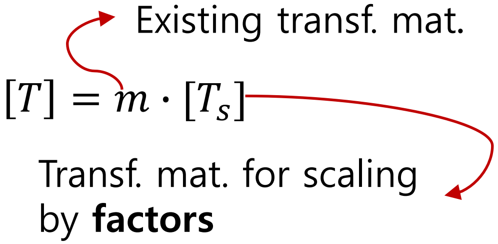


```
Ry = (0.707107 0.000000 -0.707107 0.000000)
     (0.000000 1.000000 0.000000 0.000000)
     (0.707107 0.000000 0.707107 0.000000)
     (0.000000 0.000000 0.000000 1.000000)
```

Review Transformation Functions

- Scaling
  - Required Header file: **<glm/gtc/matrix_transform.hpp>**

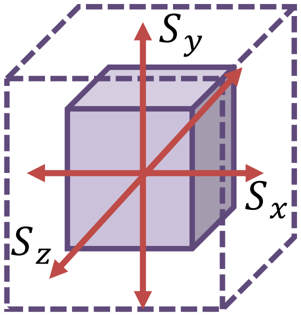
```
mat4 scale(mat4 const & m, vec3 const & factors);
dmat4 scale(dmat4 const & m, dvec3 const & factors);
```



Existing transf. mat.

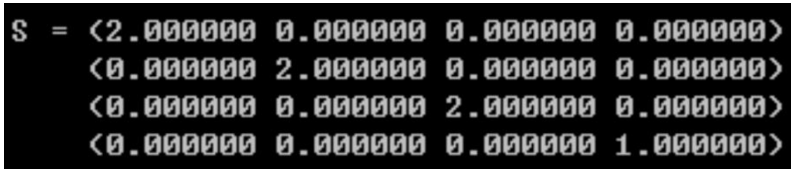
$$[T] = m \cdot [T_s]$$

Transf. mat. for scaling by **factors**



```
mat4 S(1.0f);
S = scale(S, vec3(2.0f));
printf(“ S = (%f %f %f %f)\n”, S[0][0], S[0][1], S[0][2], S[0][3]);
printf(“     (%f %f %f %f)\n”, S[1][0], S[1][1], S[1][2], S[1][3]);
printf(“     (%f %f %f %f)\n”, S[2][0], S[2][1], S[2][2], S[2][3]);
printf(“     (%f %f %f %f)\n”, S[3][0], S[3][1], S[3][2], S[3][3]);
```

$$S(S_x, S_y, S_z) = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



```
S = (2.000000 0.000000 0.000000 0.000000)
    (0.000000 2.000000 0.000000 0.000000)
    (0.000000 0.000000 2.000000 0.000000)
    (0.000000 0.000000 0.000000 1.000000)
```

Review Transformation Functions

- Translation
  - Required Header file: **<glm/gtc/matrix_transform.hpp>**

mat4 **translate**(mat4 const & **m**, vec3 const & **translation**);

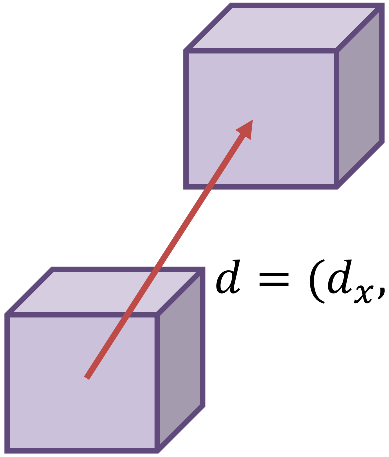
dmat4 **translate**(dmat4 const & **m**, dvec3 const & **translation**);

Existing transf. mat.

$$[T] = m \cdot [T_t]$$

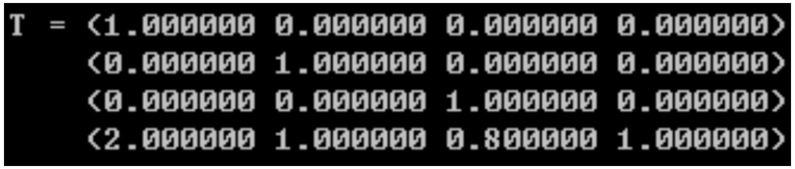
Transf. mat. for translation by **translation**

```
mat4 T(1.0f);
T = translate(T, vec3(2.0f, 1.0f, 0.8f));
printf(“ T = (%f %f %f %f)\n”, T[0][0], T[0][1], T[0][2], T[0][3]);
printf(“     (%f %f %f %f)\n”, T[1][0], T[1][1], T[1][2], T[1][3]);
printf(“     (%f %f %f %f)\n”, T[2][0], T[2][1], T[2][2], T[2][3]);
printf(“     (%f %f %f %f)\n”, T[3][0], T[3][1], T[3][2], T[3][3]);
```



$$d = (d_x, d_y, d_z)^T$$

$$T(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



```
T = (1.000000 0.000000 0.000000 0.000000)
    (0.000000 1.000000 0.000000 0.000000)
    (0.000000 0.000000 1.000000 0.000000)
    (2.000000 1.000000 0.800000 1.000000)
```

How to rotate the models

In main.cpp

```
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
#include <time.h>
```

It defines function value_ptr(…)
Which returns the pointer of
a given vector or matrix.

We will use function clock() for
a simple animation

How to rotate the models

In main.cpp

```
void display()
{
        ... codes ...

        mat4 Transf(1.0f);
        GLfloat theta = 0.001f * clock();
        Transf = rotate(Transf, theta, vec3(-1.0f, 1.0f, 0.0f));
        Transf = scale(Transf, vec3(1.0f));
        glUniformMatrix4fv(1, 1, GL_FALSE, value_ptr(Transf));

        ... codes ...
        glutSwapBuffers(); // ==glutPostRedisplay
}
```

$$T = I \cdot [T_r]$$
$$= [T_r]$$

$$T = T \cdot [T_s]$$

How to rotate the models

In main.cpp

```
void display()
{
        … codes …

        mat4 Transf(1.0f);
        GLfloat theta = 0.001f * clock();
        Transf = rotate(Transf, theta, vec3(-1.0f, 1.0f, 0.0f));
        Transf = scale(Transf, vec3(1.0f));
        glUniformMatrix4fv(1, 1, GL_FALSE, value_ptr(Transf));

        … codes …
        glutSwapBuffers(); // ==glutPostRedisplay
}
```

Pass the matrix data in T to the uniform variable with a given ID number

# of mats.

transpose?

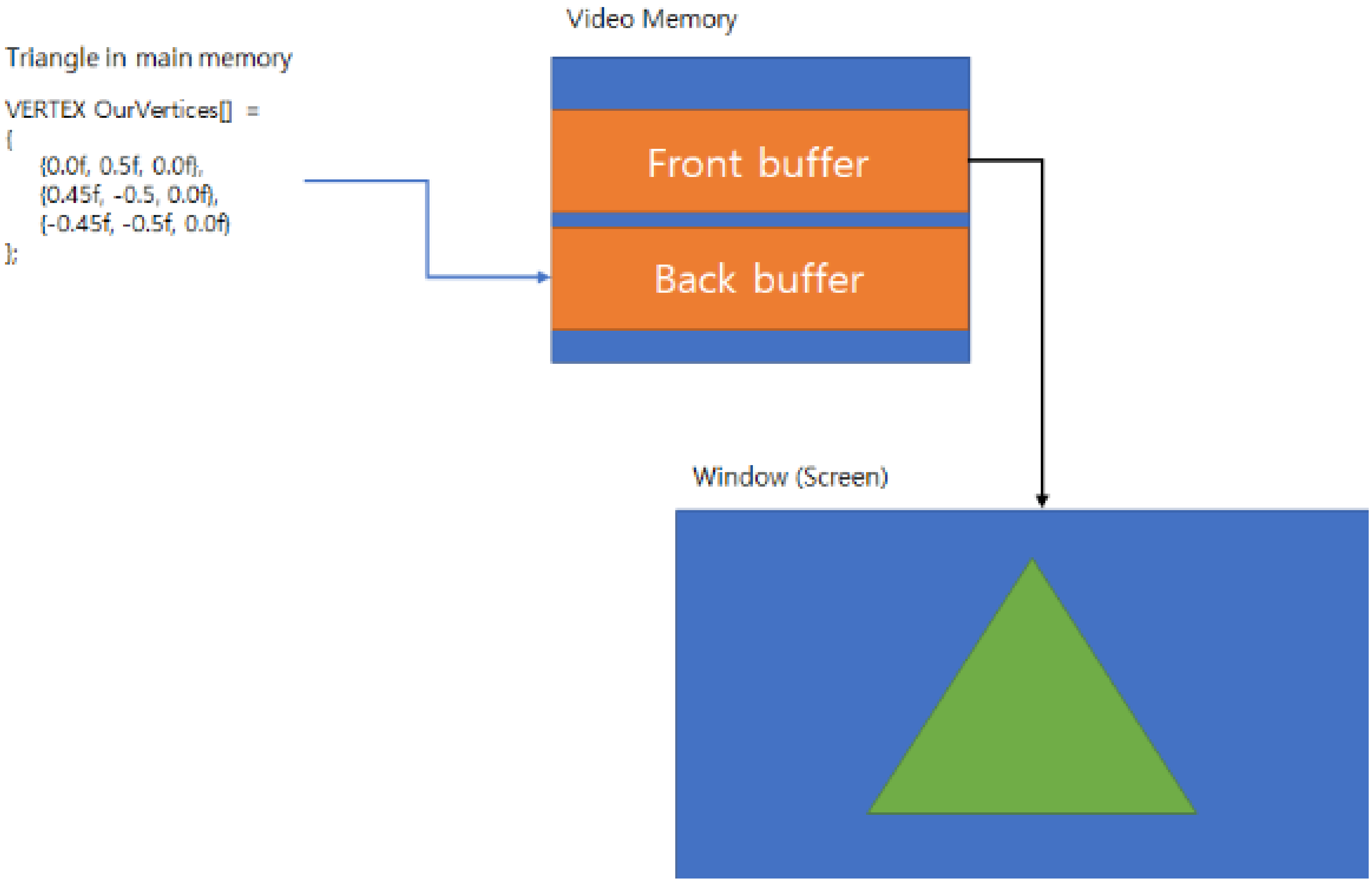
How to rotate the models

In main.cpp

```
void display()
{
        ... codes ...

        mat4 Transf(1.0f);
        GLfloat theta = 0.001f * clock();
        Transf = rotate(Transf, theta, vec3(-1.0f, 1.0f, 0.0f));
        Transf = scale(Transf, vec3(1.0f));
        glUniformMatrix4fv(1, 1, GL_FALSE, value_ptr(Transf));

        ... codes ...
        glutSwapBuffers(); // ==glutPostRedisplay
}
```

An implicit glFlush is done
by glutSwapBuffers before it returns.

How to rotate the models



Triangle in main memory

```
VERTEX OurVertices[] =
{
    {0.0f, 0.5f, 0.0f},
    {0.45f, -0.5, 0.0f},
    {-0.45f, -0.5f, 0.0f}
};
```

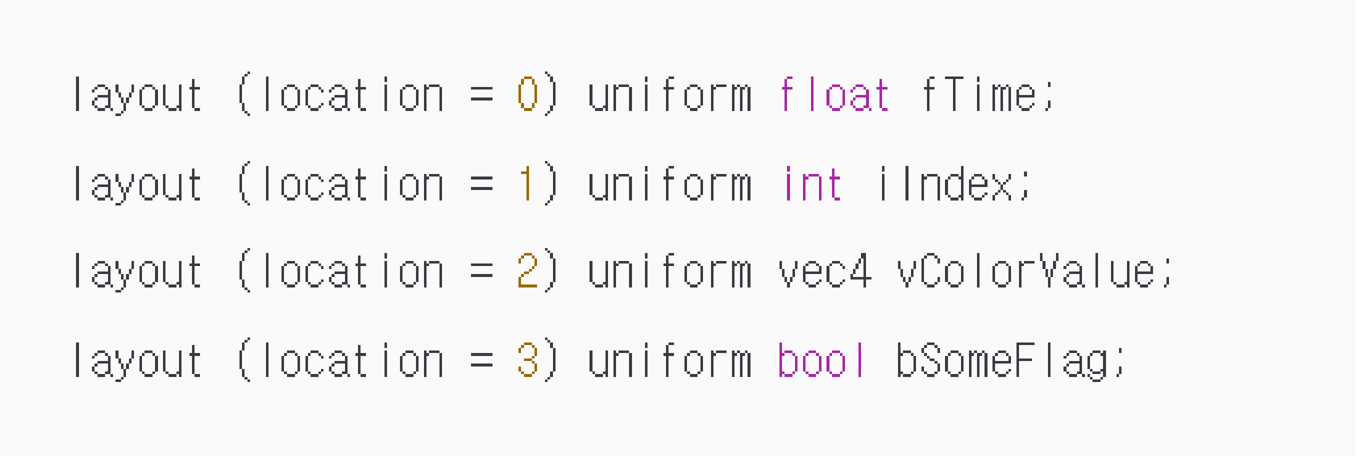
How to rotate the models

```
#version 430


in vec4 vPosition;
in vec4 vColor;
out vec4 fColor;
layout(location=1) uniform mat4 M;
layout(location=2) uniform mat4 V;
layout(location=3) uniform mat4 P;


void main()
{
    gl_Position = P * V * M * vPosition;
    fColor = vColor;
}
```

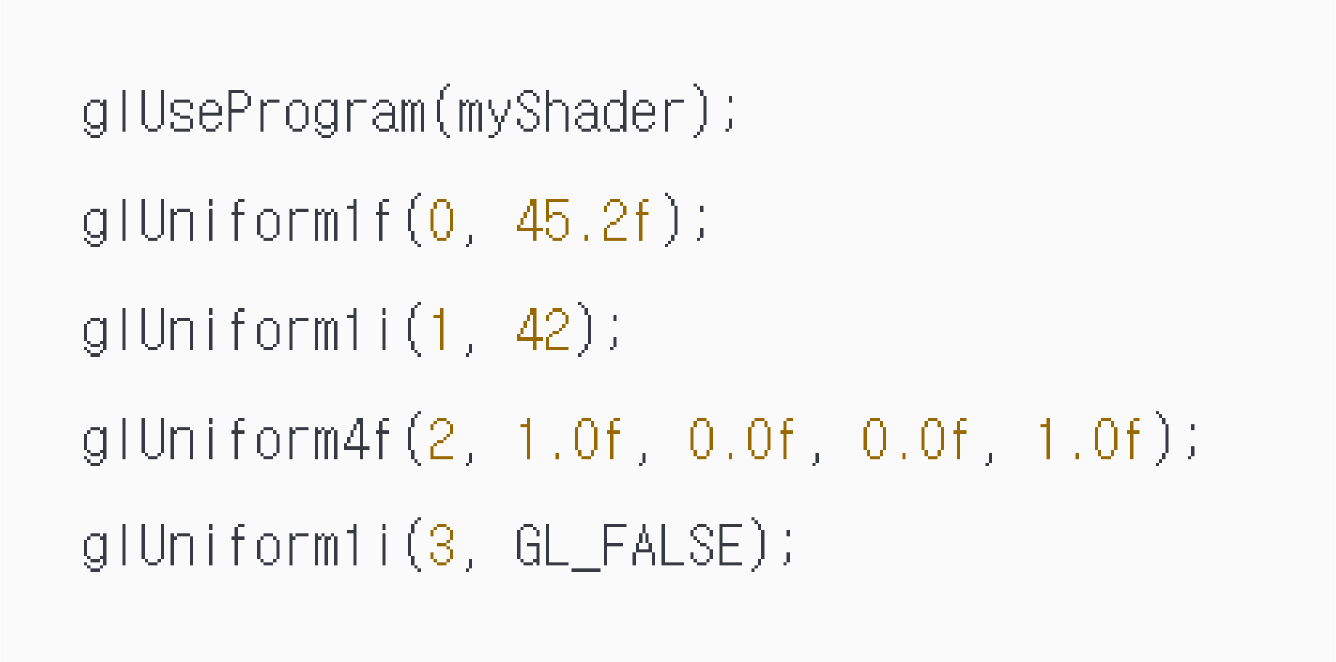
value_ptr(Transf)

A uniform variable is a global Shader variable, whose value **does not change** from one shader to the next within a particular **rendering call**.

How to rotate the models

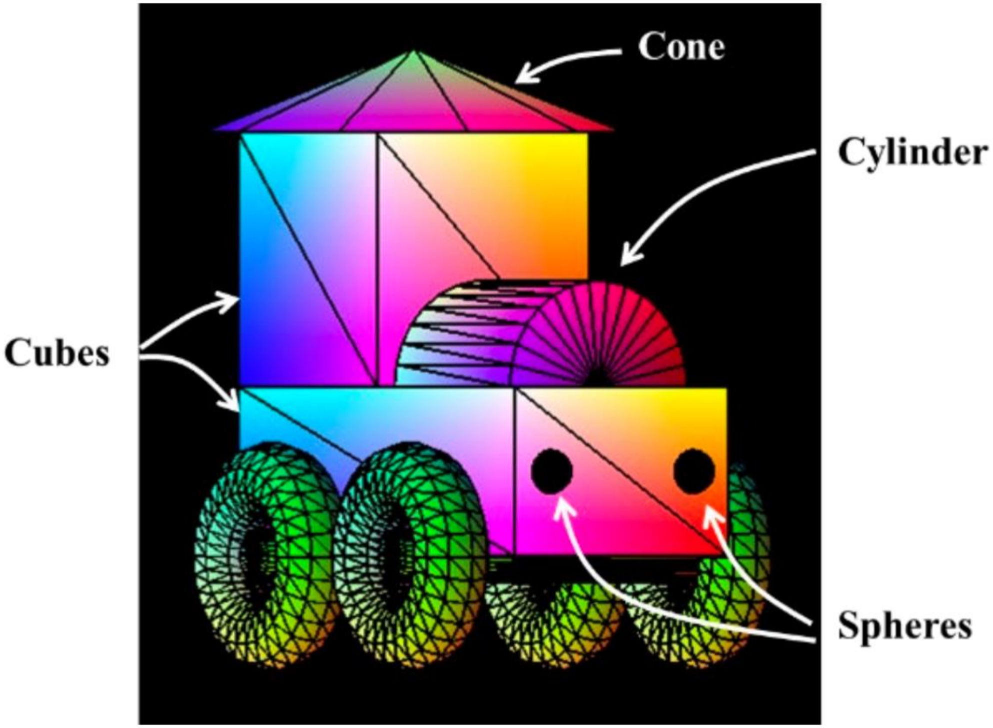


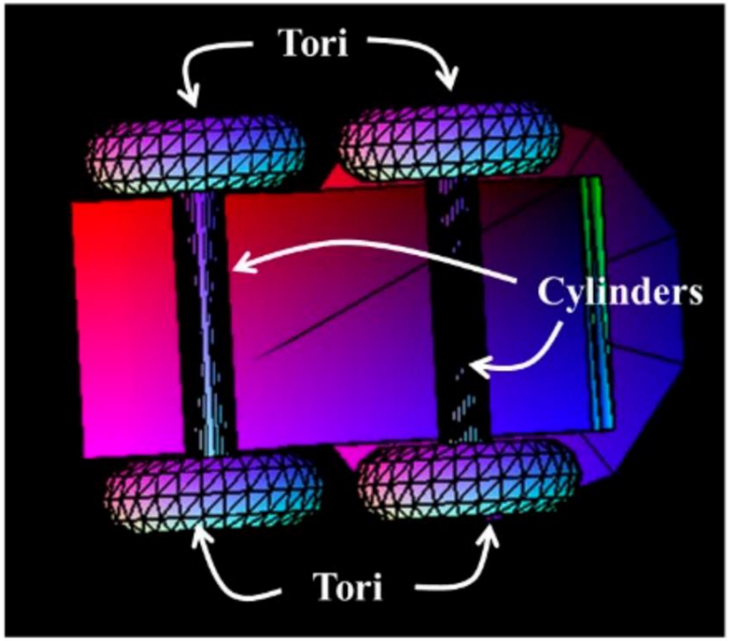
```
layout (location = 0) uniform float fTime;
layout (location = 1) uniform int iIndex;
layout (location = 2) uniform vec4 vColorValue;
layout (location = 3) uniform bool bSomeFlag;
```



```
glUseProgram(myShader);
glUniform1f(0, 45.2f);
glUniform1i(1, 42);
glUniform4f(2, 1.0f, 0.0f, 0.0f, 1.0f);
glUniform1i(3, GL_FALSE);
```

How to draw the composite model





How to draw the composite model

```
In Primi.h
Struct CarModel:public Model
{
    … codes …
    glm::mat4 transf(
                    GLfloat sx, GLfloat sy, GLfloat sz,
                    GLfloat tx, GLfloat ty, GLfloat tz,
                    glm::mat4*T_pre = NULL,
                    glm::mat4*T_post = NULL,
                    bool set_uniform = true)
    {
        using namespace glm;
        glm::mat4 T;
        T = translate(T, vec3(tx, ty, tz));
        T = scale(T, vec3(sx, sy, sz));
        if (T_pre) T = (*T_pre)*T;
        if (T_post) T = T*(*T_post);
        if (set_uniform){
            glUniformMatrix4fv(1, 1, GL_FALSE, value_ptr(T));
        }
        return T;
    }
    … codes …
```

This "transf(…)" makes matrix multiplication more convenient.

$T(t_x, t_y, t_z)$

$T(t_x, t_y, t_z) \cdot S(s_x, s_y, s_z)$

How to draw the composite model

In Primi.h

```
Struct CarModel:public Model
{
    … codes …
    glm::mat4 transf(
                    GLfloat sx, GLfloat sy, GLfloat sz,
                    GLfloat tx, GLfloat ty, GLfloat tz,
                    glm::mat4*T_pre = NULL,
                    glm::mat4*T_post = NULL,
                    bool set_uniform = true)
    {
        using namespace glm;
        glm::mat4 T;
        T = translate(T, vec3(tx, ty, tz));
        T = scale(T, vec3(sx, sy, sz));
        if (T_pre) T = (*T_pre)*T;
        if (T_post) T = T*(*T_post);
        if (set_uniform){
            glUniformMatrix4fv(1, 1, GL_FALSE, value_ptr(T));
        }
        return T;
    }
    … codes …
```

*Remind: Due to the **noncommutativity** of the multiplication operation on matrices, the order in which the matrices occur matters

How to draw the composite model

In Primi.h

```
    virtual void draw()
    {
        using namespace glm;

        GLfloat theta = 0.001f * clock();
        mat4 Rz = rotate(mat4(), -2*theta, vec3(0.0f, 0.0f, 1.0f));
        mat4 Ry = rotate(mat4(), -theta, vec3(0.0f, 1.0f, 0.0f));

        // car main body
        transf(1.2f, 0.4f, 0.6f, +0.0f, -0.2f, 0.0f, &Ry);
        cube->draw();

    … codes …
```

For a simple animation:
Rz used for rotation of the wheels.(about Z-axis)
Ry used for rotation of the whole model.(about Y-axis)

How to draw the composite model

In Primi.h

```
    virtual void draw()
    {
        using namespace glm;

        GLfloat theta = 0.001f * clock();
        mat4 Rz = rotate(mat4(), -2*theta, vec3(0.0f, 0.0f, 1.0f));
        mat4 Ry = rotate(mat4(), -theta, vec3(0.0f, 1.0f, 0.0f));

        // car main body
        transf(1.2f, 0.4f, 0.6f, +0.0f, -0.2f, 0.0f, &Ry);
        cube->draw();

    … codes …
```

Scaling factors

Translation factors

$$R_y \cdot T(0, -0.2, 0) \cdot S(1.2, 0.4, 0.6)$$

How to draw the composite model

... codes ...

```
// car upper body
transf(0.6f, 0.6f, 0.6f, -0.3f, +0.3f, 0.0f, &Ry);
cube->draw();
```

$R_y \cdot T(-0.3, 0.3, 0) \cdot S(0.6, 0.6, 0.6)$

```
// car front body
mat4 R_fb = rotate(mat4(), radius(90.0f), vec3(0, 0, 1)) * Ry;
transf(0.5f, 0.5f, 0.5f, +0.25f, 0.0f, 0.0f, &Ry, &R_fb);
cylinder->draw();
```

... codes ...

How to draw the composite model

... codes ...

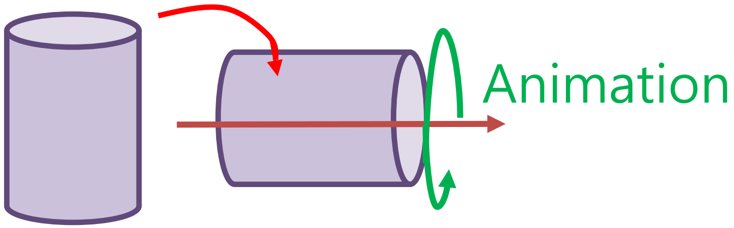
```
// car upper body
transf(0.6f, 0.6f, 0.6f, -0.3f, +0.3f, 0.0f, &Ry);
cube->draw();
```

$R_z(90°) \cdot R_y$

```
// car front body
mat4 R_fb = rotate(mat4(1.0f), radius(90.0f), vec3(0, 0, 1)) * Ry;
transf(0.5f, 0.5f, 0.5f, +0.25f, 0.0f, 0.0f, &Ry, &R_fb);
cylinder->draw();
```

... codes ...

$$R_y \cdot T(0.25,0,0) \cdot S(0.5,0.5,0.5) \cdot R_z(90°) \cdot R_y$$



Position : (0.25, 0, 0)
Size : X0.5

How to draw the composite model

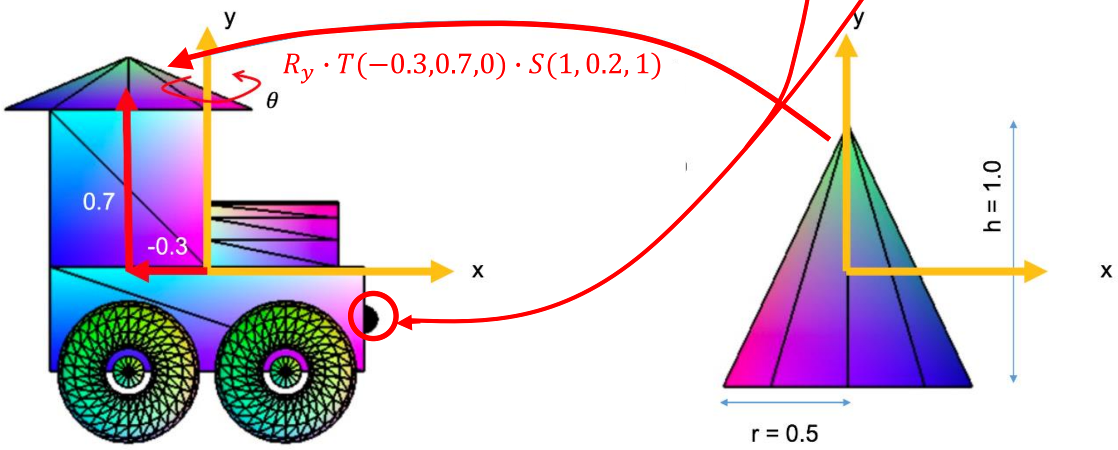
… codes …

```
// car roof
transf(1.0f, 0.2f, 1.0f, -0.3f, +0.7f, 0.0f, &Ry);
cone->draw();

// car front-right light
transf(0.1f, 0.1f, 0.1f, +0.6f, -0.2f, -0.2f &Ry);
sphere->draw();
// car front-left light
transf(0.1f, 0.1f, 0.1f, +0.6f, -0.2f, +0.2f, &Ry);
sphere->draw();
```

$R_y \cdot T(0.6, -0.2, \boxed{-0.2}) \cdot S(0.1, 0.1, 0.1)$

$R_y \cdot T(0.6, -0.2 \boxed{0.2}) \cdot S(0.1, 0.1, 0.1)$

… codes …



$R_y \cdot T(-0.3, 0.7, 0) \cdot S(1, 0.2, 1)$

How to draw the composite model

... codes ...

```
// fornt left tire
mat4 R_tire = Rz * rotate(mat4(1.0f), radians(90.0f), vec3(1, 0, 0));
transf(0.3f, 0.3f, 0.3f, +0.3f -0.4f, -0.4f, &Ry, &R_tire);
torus->draw();


// fornt rightt tire
transf(0.3f, 0.3f, 0.3f, +0.3f, -0.4f, +0.4f, &Ry, &R_tire);
torus->draw();

// rear left tire
transf(0.3f, 0.3f, 0.3f, -0.3f, -0.4f, -0.4f, &Ry, &R_tire);
torus->draw();


// rear right tire
transf(0.3f, 0.3f, 0.3f, -0.3f, -0.4f, +0.4f, &Ry, &R_tire);
torus->draw();

// front shaft
mat4 R_shaft = Rz * rotate(mat4(1.0f), radians(90.0f), vec3(1, 0, 0));
transf(0.12f, 0.12f, 0.9f,  +0.3f, -0.4f, +0.0f, &Ry, &R_shaft);
cylinder->draw();

// rear shaft
transf(0.12f, 0.12f, 0.9f,  -0.3f, -0.4f, +0.0f, &Ry, &R_shaft);
cylinder->draw();
```

$R_y \cdot T(0.3, -0.4, -0.4) \cdot S(0.3, 0.3, 0.3) \cdot R_{tire}$

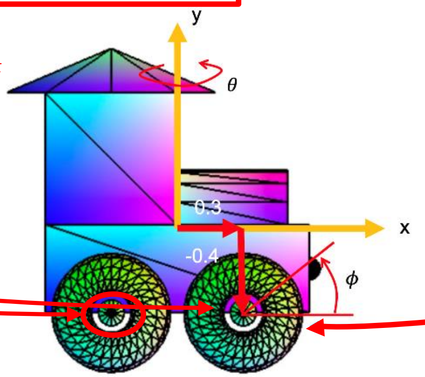
$R_y \cdot T(0.3, -0.4, 0.4) \cdot S(0.3, 0.3, 0.3) \cdot R_{tire}$

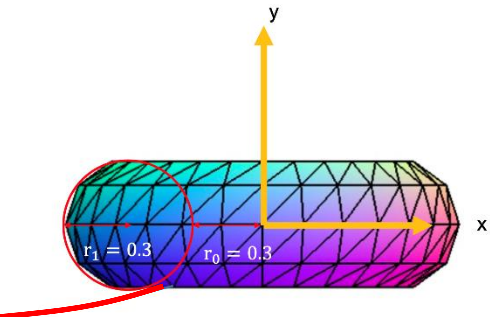
$R_y \cdot T(-0.3, -0.4, -0.4) \cdot S(0.3, 0.3, 0.3) \cdot R_{tire}$

$R_y \cdot T(-0.3, -0.4, 0.4) \cdot S(0.3, 0.3, 0.3) \cdot R_{tire}$

$R_y \cdot T(0.3, -0.4, 0) \cdot S(0.12, 0.12, 0.9) \cdot R_{shaft}$

... codes ... $R_y \cdot T(-0.3, -0.4, 0) \cdot S(0.12, 0.12, 0.9) \cdot R_{shaft}$


y
θ
0.3
x
-0.4
ϕ


y
x
$r_1 = 0.3$
$r_0 = 0.3$