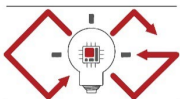


Lecture slides (AI3501/CT4201/EC4215 – Computer Graphics)

OpenGL: Setup 3D World

Lecturer: Bochang Moon



Computer Graphics
Laboratory

Prerequisite for 3D World

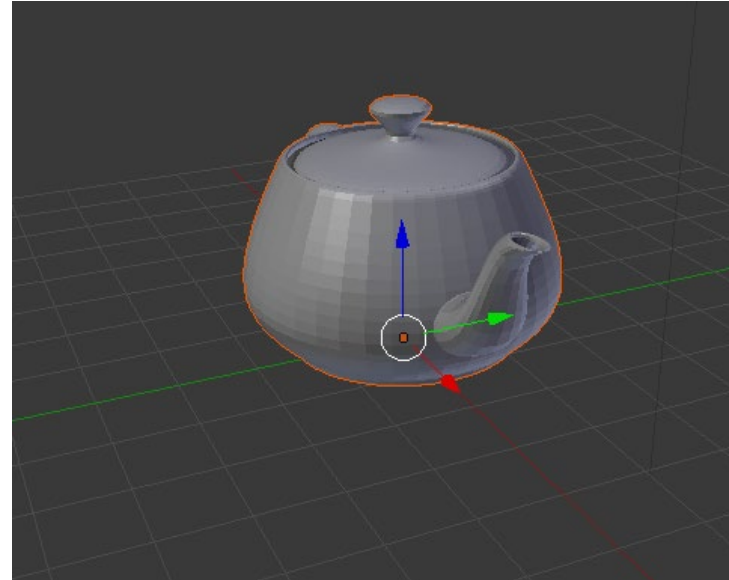
- Understanding on basic mathematical background, transformations, and spaces
 - Pixels, raster image, ...
 - Vector, matrix, ...
 - Model, viewing, and projection transformations
 - Object, world, eye, canonical view, and screen space

3D Model

- Definition of a model (or object) in 3D
 - Vertex
 - Normal (optional)
 - Q. Why do we need to use the vertex normal?
 - Texture coordinates (optional)
 - Face (usually triangles)
 - etc.
- File format for 3D models
 - You can make your own format only for your program.
 - Common formats
 - 3DS, MAX, ply (Stanford graphics lab), obj (Wavefront), etc.
 - Simple formats
 - ply and obj are quite simple formats
 - In this course, we will use “obj” format as this can be used in most rendering engines.

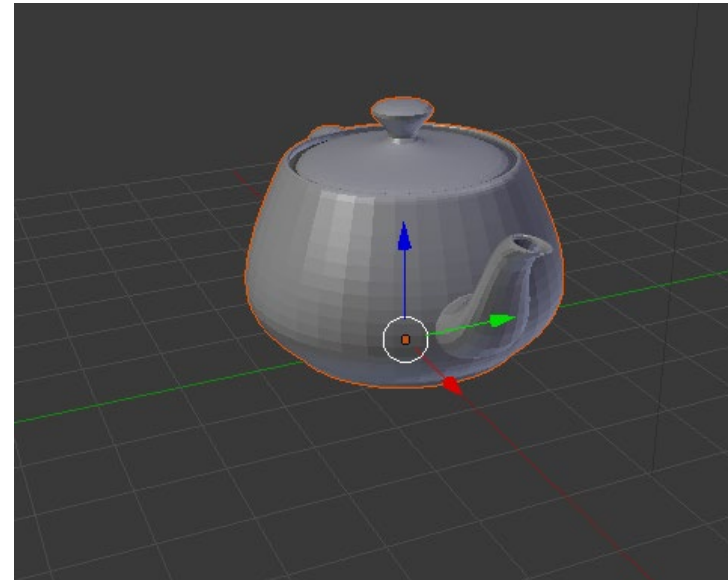
Example: .obj File Format

- Artists (or you) can design 3D models in some modeling tools (e.g., blender).
 - Out of scope...
- Most modeling tools allow us to store the models in .obj format.
 - For your homework, .obj files will be given.



Example: .obj File Format

```
v -3.000000 1.800000 0.000000      f 2909 2921 2939
v -2.991600 1.800000 -0.081000     f 2939 2931 2909
v -2.991600 1.800000 0.081000     f 2869 2877 2921
v -2.989450 1.666162 0.000000     f 2921 2909 2869
v -2.985000 1.921950 0.000000     f 2819 2827 2877
v -2.985000 1.921950 0.000000     f 2877 2869 2819
v -2.981175 1.667844 -0.081000     f 2737 2747 2827
v -2.981175 1.667844 0.081000     f 2827 2819 2737
v -2.976687 1.920243 -0.081000     f 2669 2673 2747
v -2.976687 1.920243 0.081000     f 2747 2737 2669
v -2.968800 1.800000 -0.144000     f 2567 2575 2673
v -2.968800 1.800000 0.144000     f 2673 2669 2567
v -2.958713 1.672406 -0.144000     f 2476 2480 2575
v -2.958713 1.672406 0.144000     f 2575 2567 2476
v -2.957600 1.534800 0.000000     f 2358 2362 2480
v -2.957600 1.534800 0.000000     f 2480 2476 2358
v -2.954122 1.915609 -0.144000     f 2158 2162 2362
v -2.954122 1.915609 0.144000     f 2362 2358 2158
v -2.949693 1.537790 -0.081000     f 1715 1812 2162
v -2.949693 1.537790 0.081000     f 2162 2158 1715
v -2.940000 2.019600 0.000000     f 2901 2909 2931
v -2.935200 1.800000 -0.189000     f 2931 2917 2901
v -2.935200 1.800000 0.189000     f 2863 2869 2909
v -2.931958 2.016526 0.081000     f 2909 2901 2863
v -2.931958 2.016526 -0.081000    f 2813 2819 2869
v -2.928230 1.545907 -0.144000     f 2869 2863 2813
v -2.928230 1.545907 0.144000     f 2729 2737 2819
v -2.925611 1.679131 -0.189000     f 2819 2813 2729
v -2.925611 1.679131 0.189000     f 2663 2669 2737
v -2.920870 1.908779 -0.189000     f 2737 2729 2663
v -2.920870 1.908779 0.189000     f 2561 2567 2669
v -2.910131 2.008181 -0.144000     f 2669 2663 2561
v -2.910131 2.008181 0.144000     f 2468 2476 2567
v -2.904150 1.406137 0.000000     f 2567 2561 2468
v -2.904150 1.406137 0.000000     f 2350 2358 2476
v -2.896846 1.410135 0.081000     f 2476 2468 2350
v -2.896846 1.410135 -0.081000    f 2152 2158 2358
v -2.896602 1.557869 -0.189000     f 2358 2350 2152
v -2.896602 1.557869 0.189000     f 1717 1715 2158
v -2.894400 1.800000 -0.216000     f 2158 2152 1717
v -2.894400 1.800000 0.216000     f 2903 2901 2917
```



teapot.obj (toy example)

- 3644 vertices
- 6320 faces

Example: .obj File Format

- #: comment line
- v x y z w
 - Vertex coordinates in model space
 - w: optional (default = 1)
- vt u v
 - Texture coordinates ($0 \leq u, v \leq 1$)
- vn x y z
 - Normal direction
- f v1 v2 v3
 - v1: index in the vertex list (integer)
- Q. why do they use the vertex index instead of coordinates?

Example: .obj File Format

- f v1 v2 v3
 - v1: index in the vertex list (integer)
- f v1/vt1 v2/vt2 v3/vt3
 - vt: texture coordinate (index)
- f v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn3
 - vn: normal (index)
- f v1//vn1 v2//vn2 v3//vn3
 - Need empty slash to avoid ambiguity

Loading .obj Model

- Read .obj files from your disk
 - Build a vertex list from each model
 - (optional) Create normal and texture coordinate lists
 - Build a face list
 - In our example, we will use triangles.
- A very simple .obj loader will be given for your assignments.
 - ModelLoader.h & ModelLoader.cpp

Draw Triangles

- For each triangle in a model
 - glBegin(GL_TRIANGLES)
 - For each vertex in a triangle
 - glVertex3d(x, y, z)
 - (optionally)
 - glNormal3d(nx, ny, nz) // related to shading
 - glTexCoord2d(u, v) // related to texture mapping
 - glEnd()

OpenGL Display List

- Display list: a set of OpenGL commands that have been stored for later execution
- Once the list is compiled (one time), it can be re-used multiple times.
 - Very efficient for static models

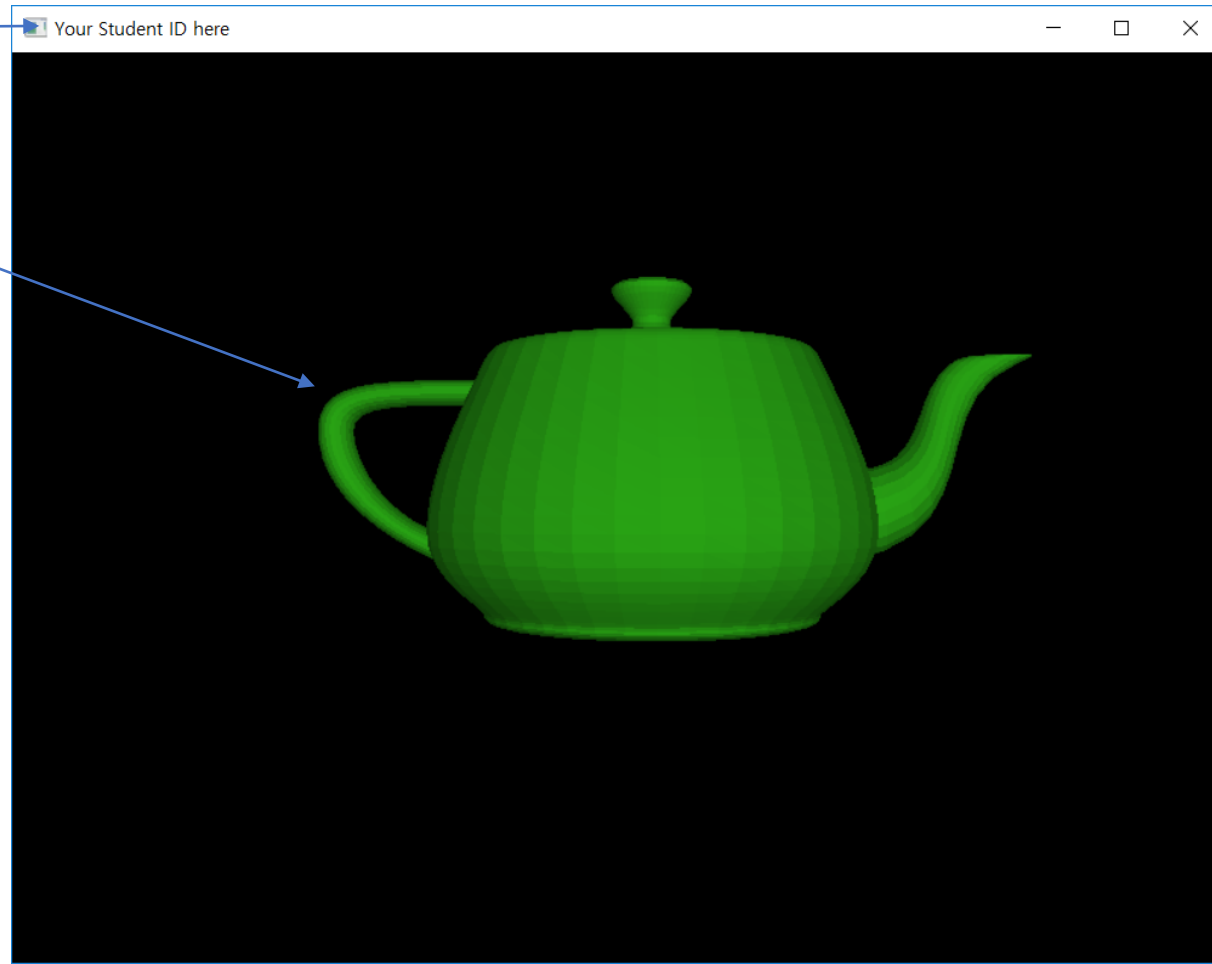
Example of OpenGL Display List

- `g_teapotID = glGenLists(1) // create a list and store the ID to the variable`
 - `glNewList(g_teapotID, GL_COMPILE) // Define the set of commands`
 - `glBegin(GL_TRIANGLES)`
 - For each vertex in a triangle
 - `glVertex3d(x, y, z)`

 - (optionally)
 - `glNormal3d(nx, ny, nz) // related to shading`
 - `glTexCoord2d(u, v) // related to texture mapping`
 - `glEnd()`
 - `glEndList()`
-
- Draw some models with IDs
 - `glCallList(g_teapotID)`

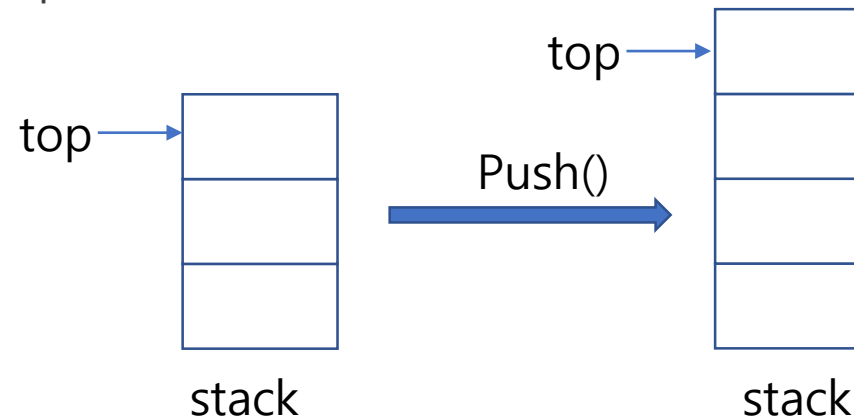
Result Image

- Title bar
- Teapot model
 - A set of triangles

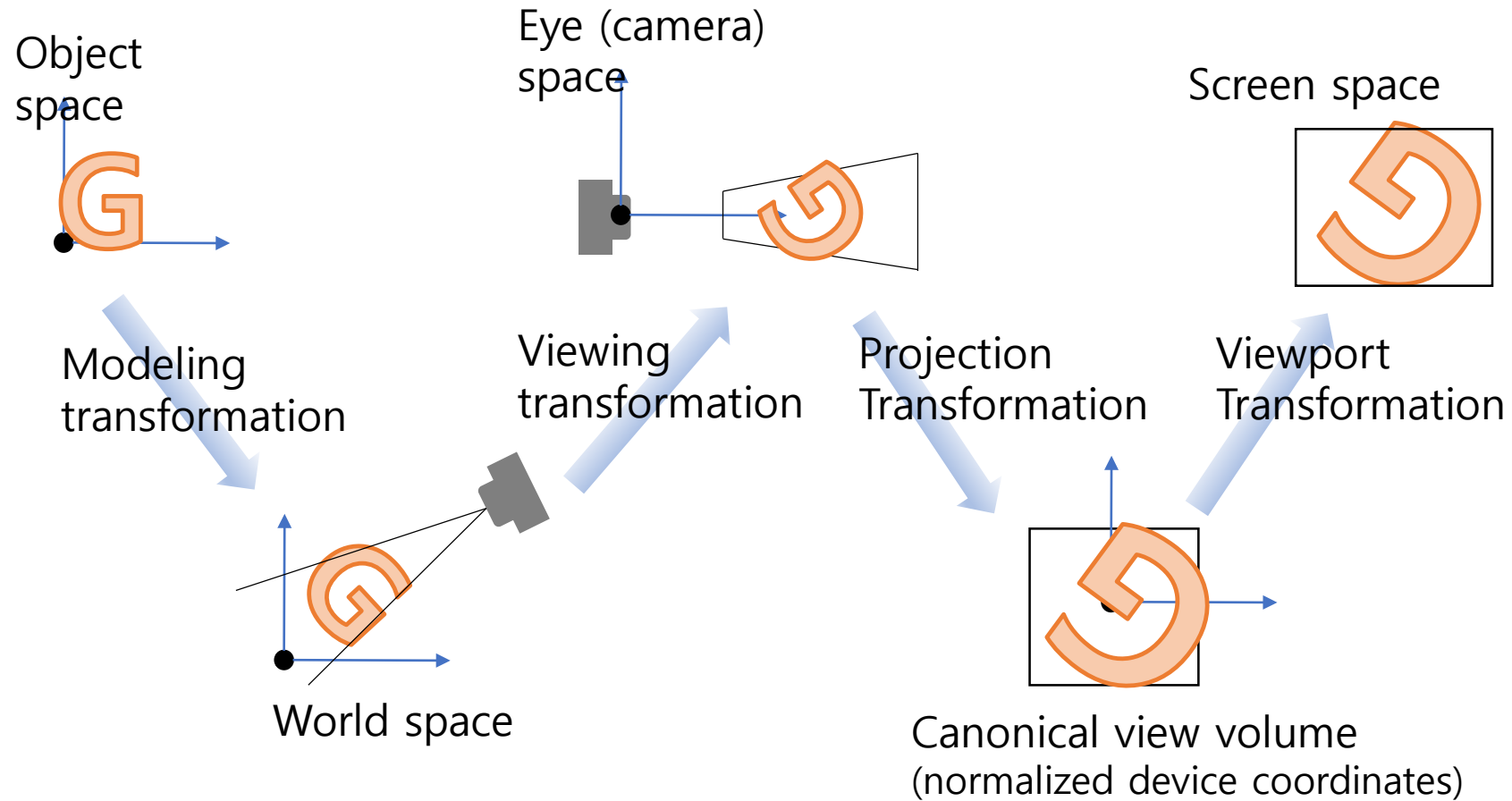


OpenGL Matrix Mode

- `glMatrixMode(mode)`
 - mode: specify which matrix stack is the target that we want to modify
 - `GL_PROJECTION`
 - `GL_MODEL_VIEW`
 - `GL_TEXTURE` (not used now)
- Stack?
 - An abstract data structure that can contain multiple elements
 - LIFO: Last In First Out
 - Interfaces
 - Push: insert a new element into the stack
 - Pop: delete an element from the stack
 - The top element can be modified.



Sequence of Spaces and Transformations



OpenGL Matrix Mode

- `glMatrixMode(mode)`
 - mode: specify which matrix stack is the target that we want to modify
 - `GL_PROJECTION`
 - `GL_MODEL_VIEW`
 - `GL_TEXTURE` (not used now)
 - `GL_PROJECTION`
 - Related to the projection transformation
 - `GL_MODEL_VIEW`
 - Related to the modeling and viewing transformation
 - Should call this function first before applying a transformation

OpenGL Matrix Mode

- void reshape(int w, int h)
 - glViewport(0, 0, w, h); // viewport transformation is specified.
 - glMatrixMode(GL_PROJECTION); // specify our target stack
 - glLoadIdentity(); // set the current matrix as the identity
 - gluPerspective(fov, aspect, near, far); // change the current matrix with the perspective transformation matrix



stack (projection)



stack (model view)

OpenGL Matrix Mode

- void reshape(int w, int h)
 - glViewport(0, 0, w, h); // viewport transformation is specified.
 - glMatrixMode(GL_PROJECTION); // specify our target stack
 - glLoadIdentity(); // set the current matrix as the identity
- Alternatively,
 - We can explicitly construct the projection matrix
 - double matrix[16] = {...}
 - glMultMatrixd(matrix); // multiply the current matrix with the specified matrix



stack (projection)



stack (model view)

OpenGL Matrix Mode

- void reshape(int w, int h)
 - glViewport(0, 0, w, h); // viewport transformation is specified.
 - glMatrixMode(GL_PROJECTION); // specify our target stack
 - glLoadIdentity(); // set the current matrix as the identity
 - gluPerspective(fov, aspect, near, far); // change the current matrix with the perspective transformation matrix
 - glMatrixMode(GL_MODELVIEW);
 - glLoadIdentity();



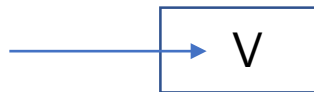
stack (projection)



stack (model view)

OpenGL Matrix

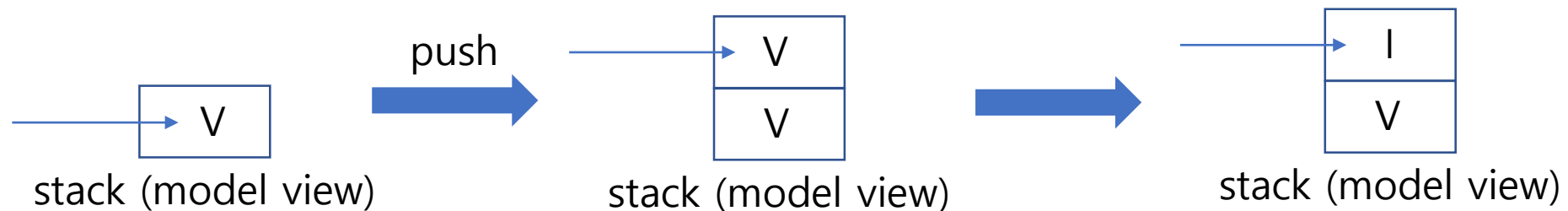
- void display()
 - `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)`
 - `setCamera()` // specify the viewing transformation (change the current matrix)
 - e.g., `gluLookAt()` or explicitly build the matrix



stack (model view)

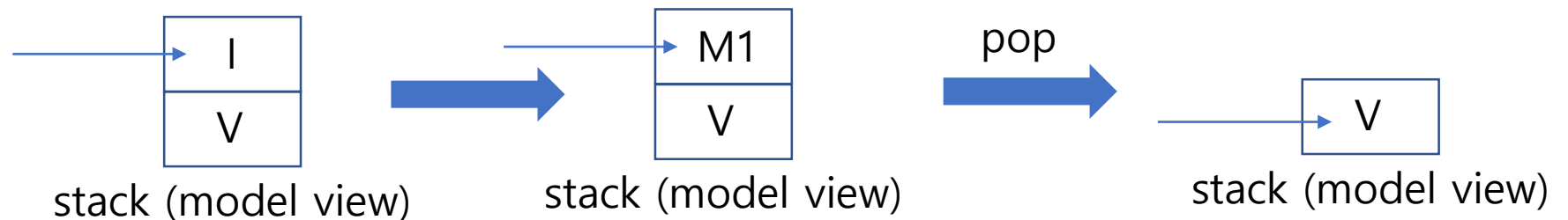
OpenGL Matrix

- For each model
 - `glPushMatrix();` // push down the current matrix, and insert a new element it to the stack (the new matrix will be equal to the current matrix - duplicated)
 - `glLoadIdentity();` // initialize the current matrix



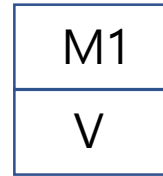
OpenGL Matrix

- For each model (continued...)
 - // specify modeling transformation matrix
 - e.g., $M = T$, $M = RT$, and etc.
 - e.g., call predefined functions (`glTranslated(...)`, `glRotated(...)`) or explicitly build the matrix and multiply it
 - e.g., `glTranslated(...)` // build the specified translation matrix and multiply it to the current matrix
 - // draw primitives
 - e.g., `glCallList(ID)` // use the display list
 - // pop the modeling transformation – why?
 - `glPopMatrix()`

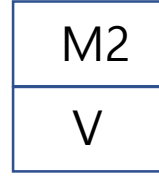


OpenGL Matrix

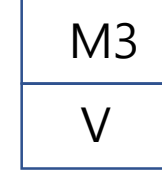
- Scenarios
 - Each model has different modeling transformations
 - The matrix stack (with different modeling matrix) should be built before drawing the primitives of each model



stack (model view)

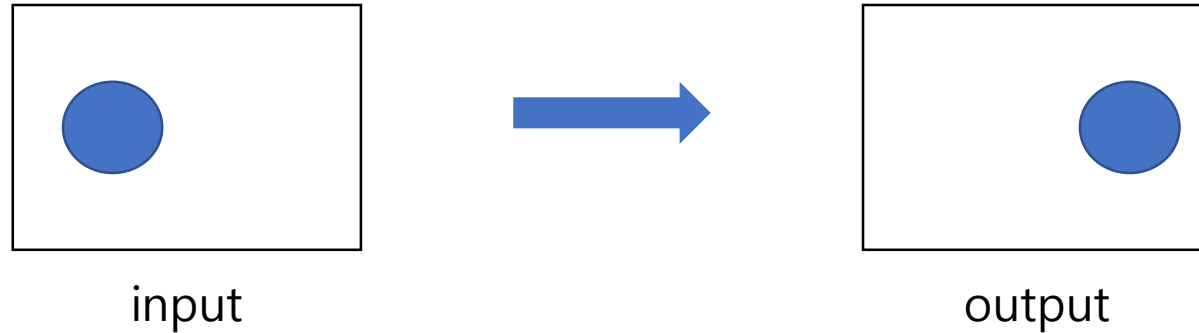


stack (model view)



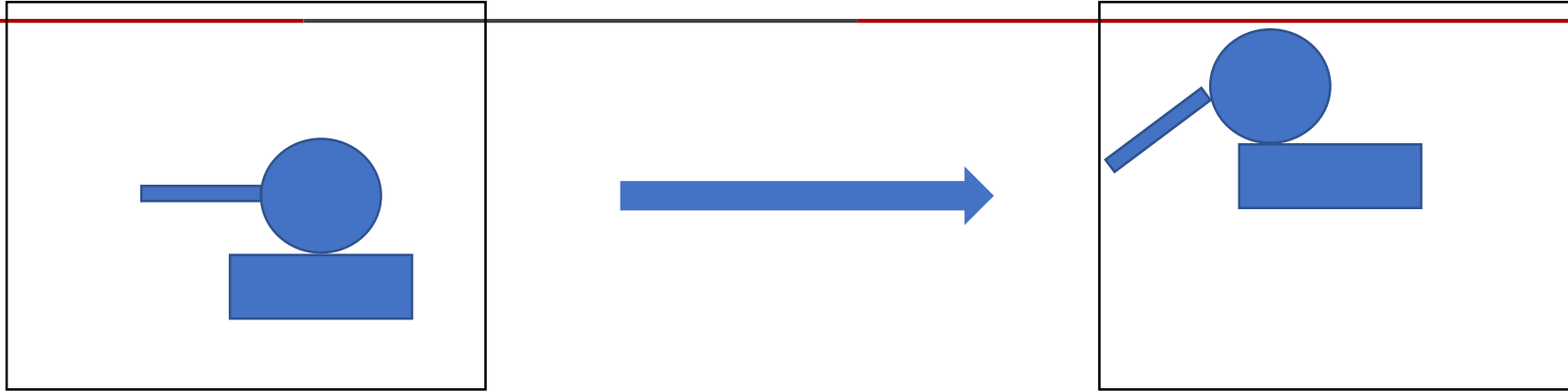
stack (model view)

Object vs Camera Transformation



- Use modeling transformation
 - Translate the object along x direction (this can be separately applied to each object)
- Use viewing transformation
 - Translate our camera along $-x$ direction (this moves the coordinate system – all objects will be transformed)

Hierarchical Transformation



Two ways of achieving this transformation:

1. Apply different transformation to each part

Translate the body

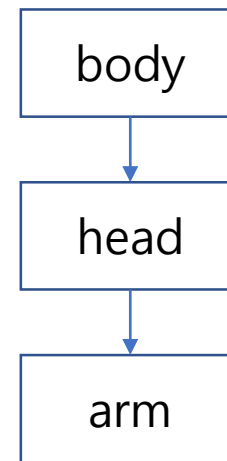
Translate the head

Rotate and translate the arm

2. Utilize relative transformation

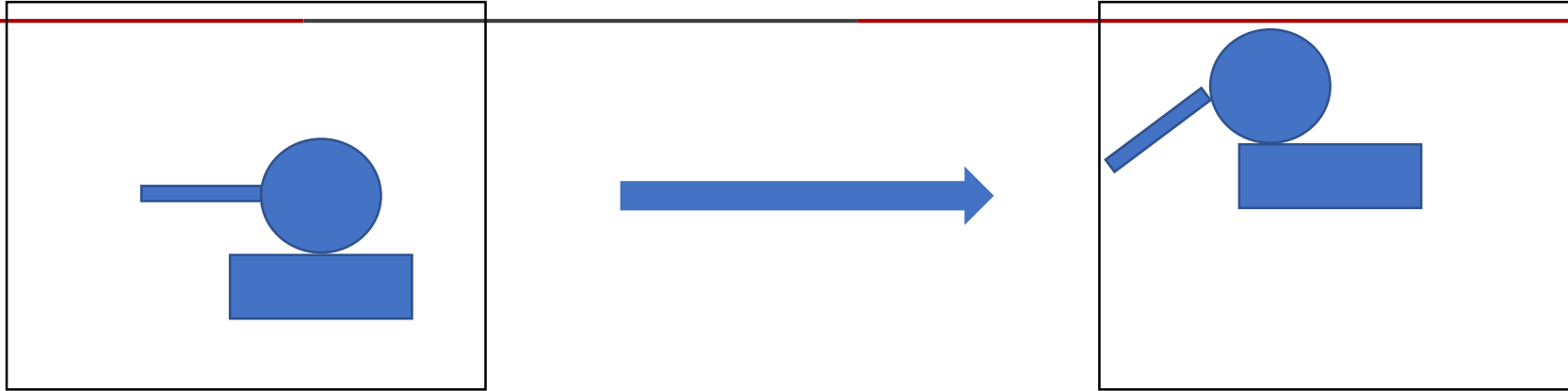
Some parts can depend on others

Specify each object's transformation relative to its parent



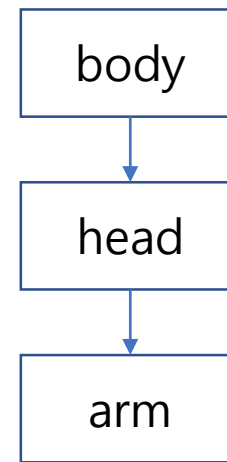
Scene graph

Hierarchical Transformation



2. Utilize relative transformation

- Translate the body and its descendants
- Translate the head and its descendant
- Rotate the arm



Scene graph

Hierarchical Transformation

2. Utilize relative transformation

```
// apply a transformation to the body and its descendants
```

```
glTranslated(...)
```

```
// draw body
```

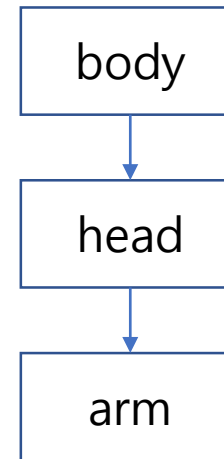
```
glTranslated(...)
```

```
// draw head
```

```
// apply a transformation to the arm
```

```
glRotated(..)
```

```
// draw arm
```



Scene graph

Hierarchical Transformation

2. Utilize relative transformation

```
// apply a transformation to the body and its descendants
```

```
glTranslated(...)
```

```
// draw body
```

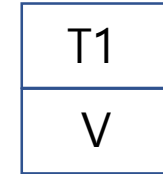
```
glTranslated(...)
```

```
// draw head
```

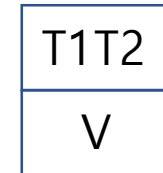
```
// apply a transformation to the arm
```

```
glRotated(..)
```

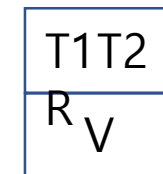
```
// draw arm
```



stack (model view)

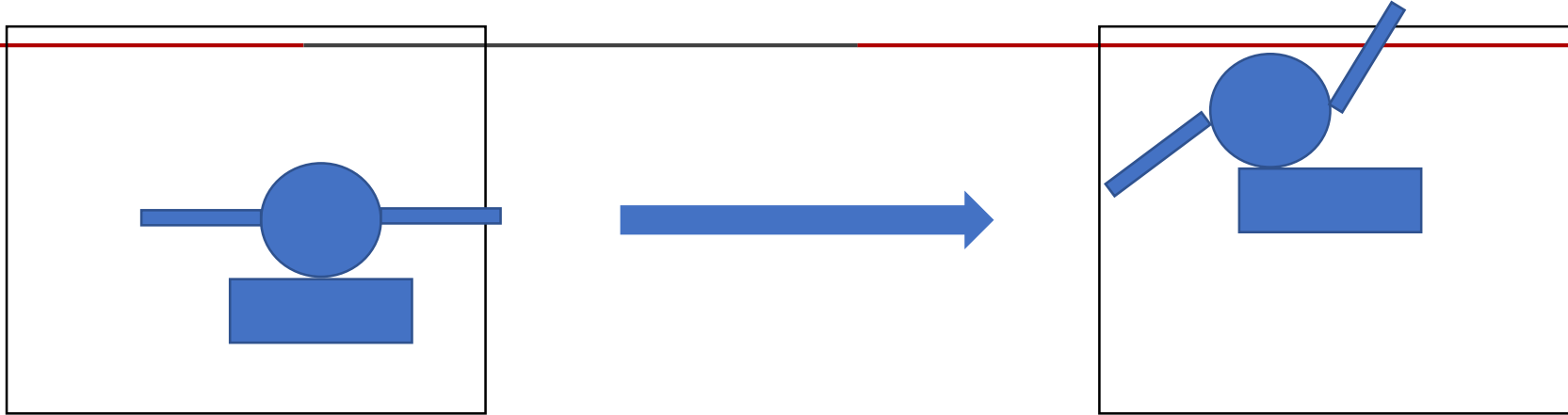


stack (model view)



stack (model view)

Hierarchical Transformation



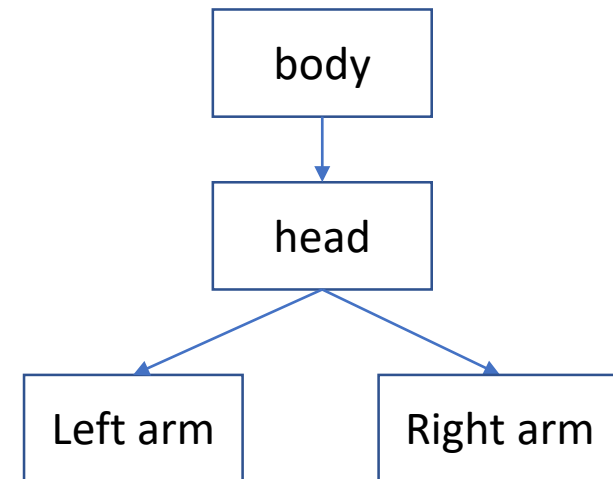
2. Utilize relative transformation

Translate the body and its descendants

Translate the head and its descendants

Rotate the left arm

Rotate the right arm?



Hierarchical Transformation

2. Utilize relative transformation

```
// apply a transformation to the body and its descendants
```

```
glTranslated(...)
```

```
// draw body
```

```
glTranslated(...)
```

```
// draw head
```

```
// apply a transformation to the left arm
```

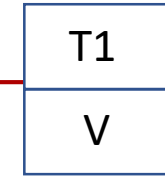
```
glRotated(..)
```

```
// draw left arm
```

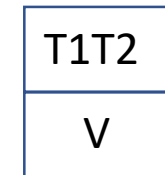
```
// apply a transformation to the right arm
```

```
glRotated(..)
```

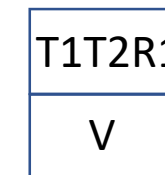
```
// draw right arm?
```



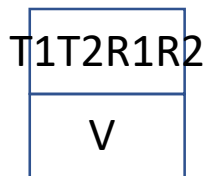
stack (model view)



stack (model view)



stack (model view)



stack (model view)

Hierarchical Transformation

2. Utilize relative transformation

```
// apply a transformation to the body and its descendants
```

```
glTranslated(...)
```

```
// draw body
```

```
glTranslated(...)
```

```
// draw head
```

```
// apply a transformation to the left arm
```

```
glRotated(..)
```

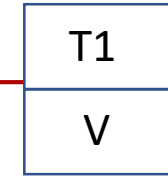
```
// draw left arm
```

```
// apply a transformation to the right arm
```

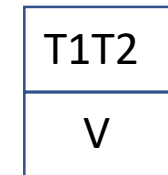
```
glRotated(..)
```

```
// draw right arm?
```

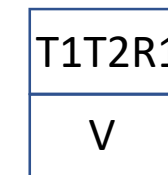
ERROR: the left arm is not a parent of the right arm



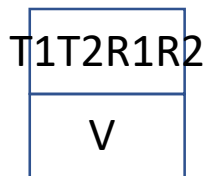
stack (model view)



stack (model view)

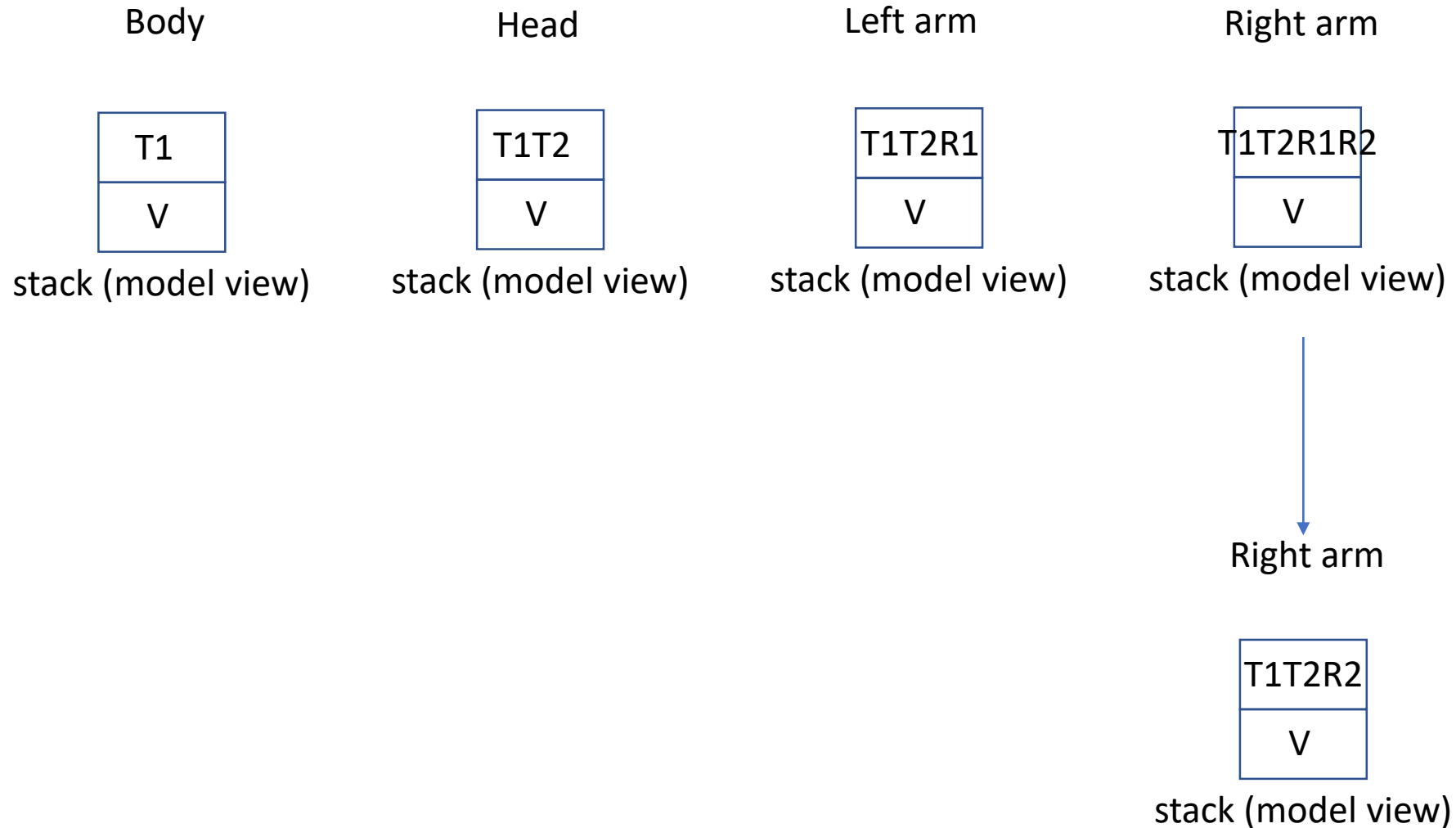


stack (model view)



stack (model view)

Hierarchical Transformation



Hierarchical Transformation

2. Utilize relative transformation

```
// apply a transformation to the body and its descendants
```

```
glTranslated(...)
```

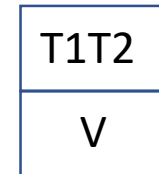
```
// draw body
```

```
glTranslated(...)
```

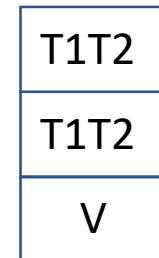
```
// draw head
```

```
// Store the current matrix!
```

```
glPushMatrix()
```



stack (model view)



stack (model view)

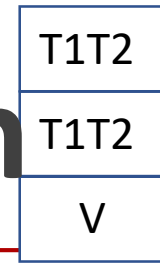
Hierarchical Transformation

```
// Store the current matrix!  
glPushMatrix()
```

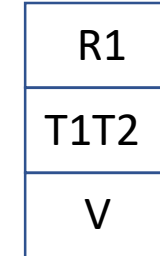
```
// apply a transformation to the left arm  
glLoadIdentity()  
glRotated(..)
```

```
// draw left arm
```

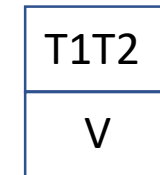
```
// Pop the top of the stack  
glPopMatrix()
```



stack (model view)



stack (model view)



stack (model view)

Hierarchical Transformation

```
// Store the current matrix!  
glPushMatrix()
```

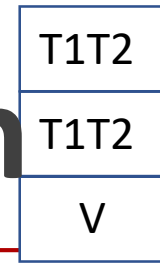
```
// apply a transformation to the left arm  
glLoadIdentity()  
glRotated(..)
```

```
// draw left arm
```

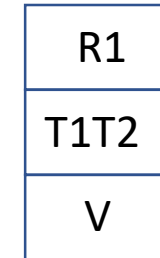
```
// Pop the top of the stack  
glPopMatrix()
```

```
// apply a transformation to the right arm  
glRotated(..)
```

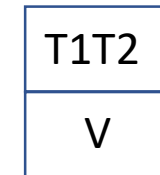
```
// draw right arm
```



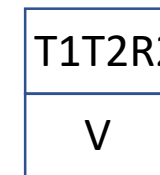
stack (model view)



stack (model view)



stack (model view)



stack (model view)