Interactive Computer Graphics

CS 418 – Spring 2011

Mesh Rendering, Transformation, Camera Viewing and Projection in OpenGL

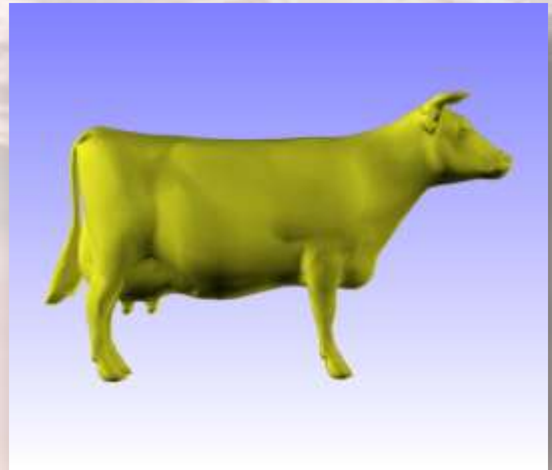
Author: Mahsa Kamali
TA: Gong Chen
Email: gchen10 at illinois.edu

Agenda

- Mesh format
- Drawing with OpenGL
- Matrix transformation
- 3 things to take home
- Gimble lock

How to Load Your Mesh

- Customized .obj 3D models **with colors**.
- Won’t work with a obj reader code.
- You should have skills to write a simple parser loading the files.



Our Mesh File Format

Position v1	v 0.0 0.0 0.0
Position v2	v 1.0 0.0 0.0
	…..
Color v1	vc 1 0 0
Color v2	vc 0 0 1
	…..
	f 1 2 3
	f 1 3 4
	…..

- You will have a list of vertex attributes : Positions (v), Colors (vc), etc.
- Vertices are indexed according to their orders in file.
- Another indexed list for triangles (f)
  - Ex : Triangle 1 is formed by vertex v1,v2 and v3

Exercise

Draw the object from the given vertex/face list :

```
v 0.0 0.0 0.0
v 1.0 0.0 0.0
v 1.0 1.0 0.0
v 0.0 1.0 0.0

vc 1 0 0
vc 0 0 1
vc 1 0 0
vc 0 1 0

f 1 2 3
f 1 3 4
```

Mesh Structure

- Face-index List :
  - Recommend to use/implement basic matrix/vector structure and operations. (ex : **libgfx**)
  - Store vertex attributes in one array
  - Store face-vertex indices in another array.
  - When rendering, iterate through each face, and grab vertex attributes based on Indices.

  - More complicated structure is possible → Half-Edge, etc.

Display Your Mesh

- Assuming you've set up the view/projection transformation.
- Display one triangle

```
glBegin(GL_TRIANGLES);
    glVertex3f(x1,y1,z1);
    glVertex3f(x2,y2,z2);
    glVertex3f(x3,y3,z3);
glEnd();
```

- glBegin→Decide which primitive you will display.
  - GL_POINTS, GL_LINES, GL_TRIANGLES, etc.
- Display a mesh is similar, just go through each triangle in the mesh.
  (Put loop between glBegin/glEnd)

Color Your Mesh

- glColor3f→Set R,G,B color
  - Range from 0.0~1.0. (1.0,1.0,1.0) is white.
- Use the provided colors, or generate your own.
- Ex : Color one triangle with Red, Green, Blue at each vertex

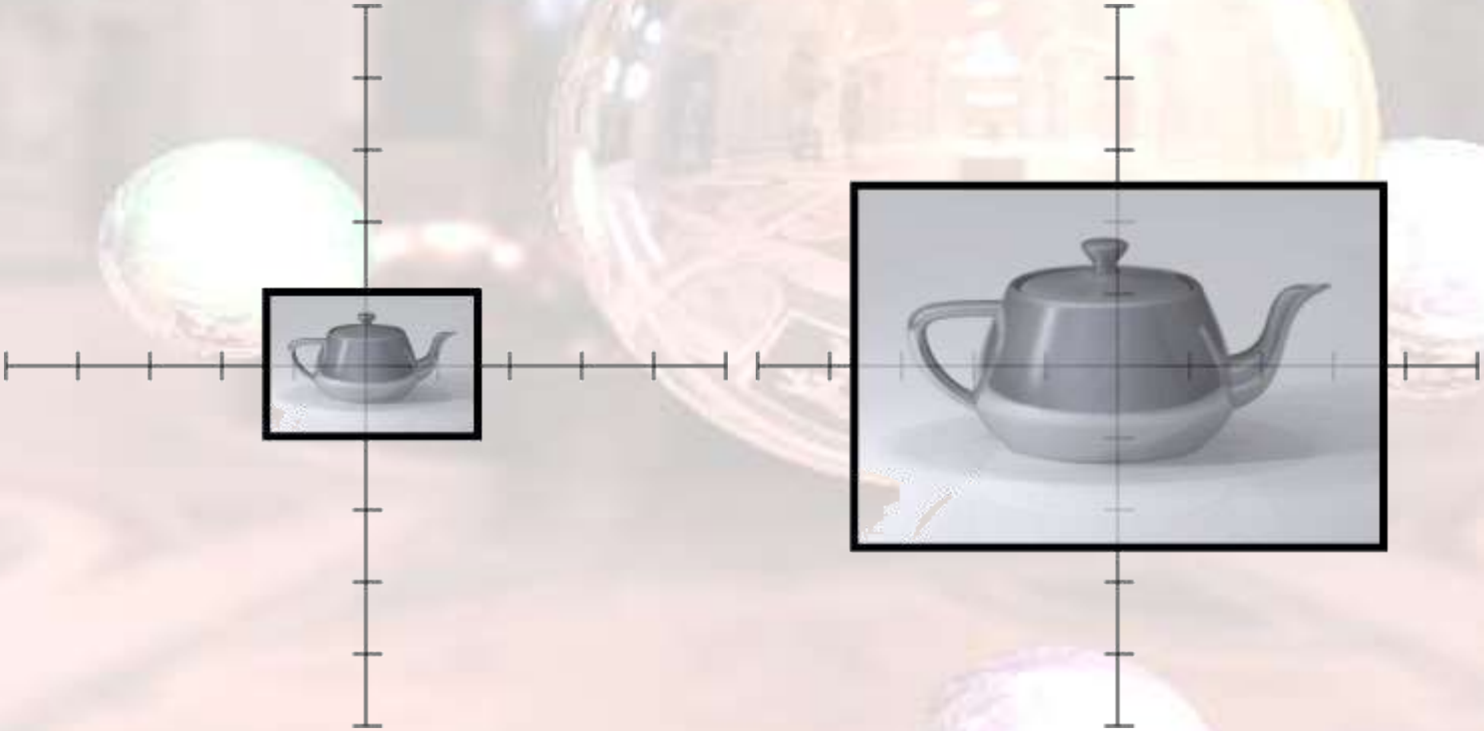
```
glBegin(GL_TRIANGLES);
    glColor3f(1.0,0.0,0.0); //red
    glVertex3f(x1,y1,z1);
    glColor3f(0.0,1.0,0.0); // green
    glVertex3f(x2,y2,z2);
    glColor3f(0.0,0.0,1.0); // blue
    glVertex3f(x3,y3,z3);
glEnd();
```

OpenGL Matrix Transformation

- Essential for interactive viewing/animation.
- Things to Take Home
  - #1 : You are modifying a global "current matrix"
  - #2 : The "last" transformation gets applied "first".
  - #3 : OpenGL store matrix in "Column Major"

Review of Matrix Ops.

- `glScalef (2.5, 2.5, 1.0);`



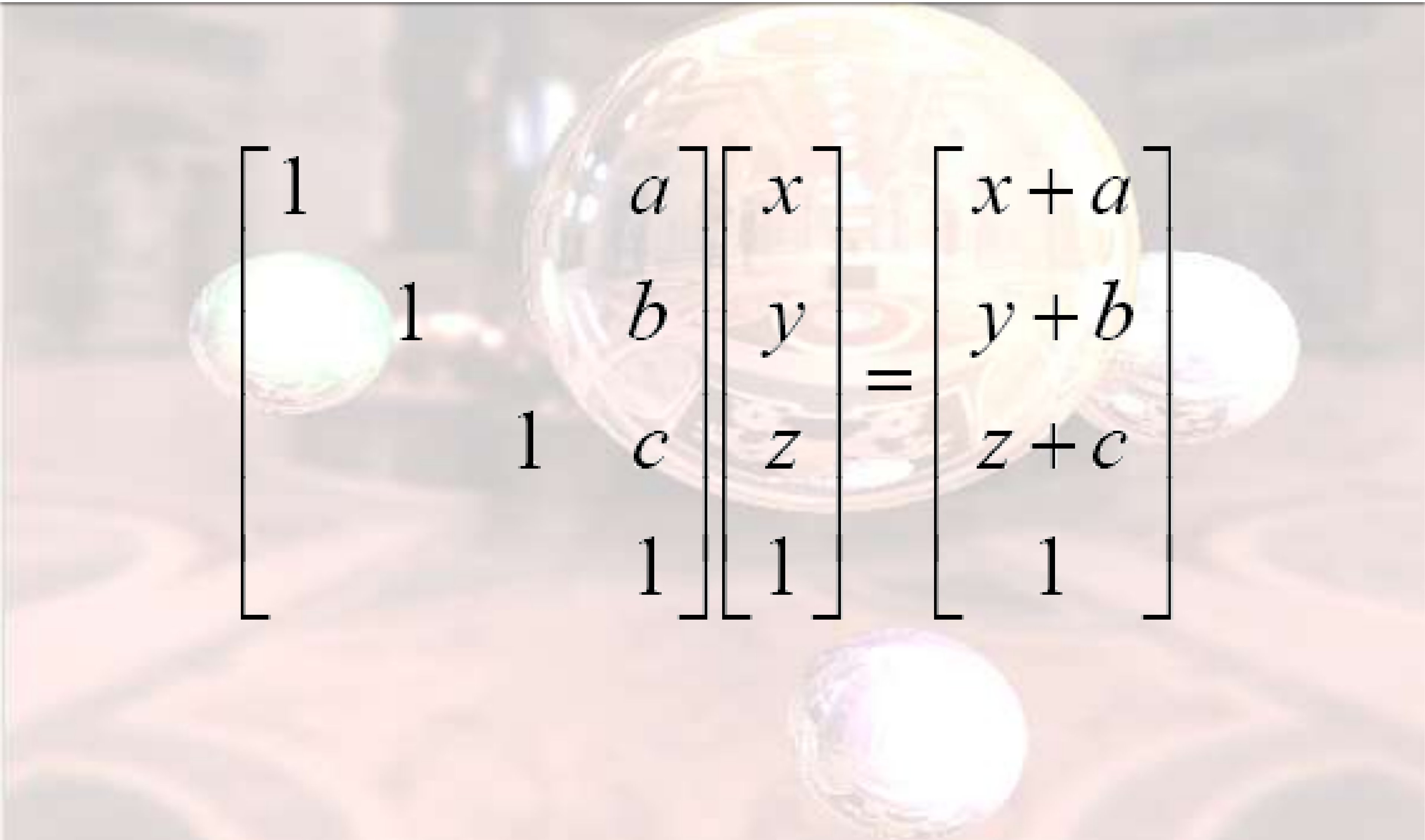
Scaling

$$\begin{bmatrix} a & & & \\ & b & & \\ & & c & \\ & & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} ax \\ by \\ cz \\ 1 \end{bmatrix}$$

Translation

- glTranslatef(2.5,2.0,0.0);

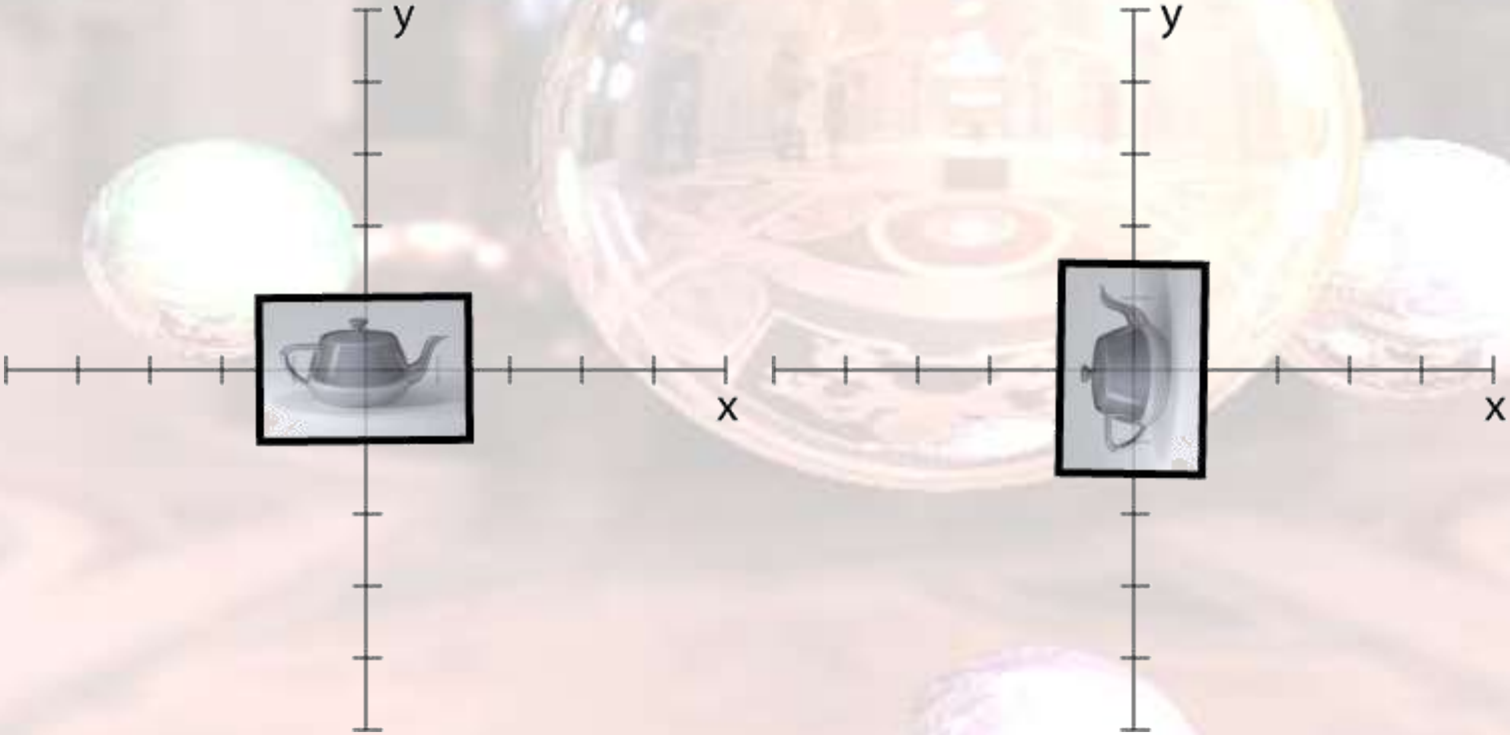
Translation



$$\begin{bmatrix} 1 & & & a \\ & 1 & & b \\ & & 1 & c \\ & & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x+a \\ y+b \\ z+c \\ 1 \end{bmatrix}$$

Rotation

- glRotatef(90.0, 0.0, 0.0, 1.0)



Rotation

- About x-axis
  - rotates $y \rightarrow z$

$$\begin{bmatrix} 1 & & & \\ & \cos\theta & -\sin\theta & \\ & \sin\theta & \cos\theta & \\ & & & 1 \end{bmatrix}$$

- About y-axis
  - rotates $z \rightarrow x$

$$\begin{bmatrix} \cos\theta & & \sin\theta & \\ & 1 & & \\ -\sin\theta & & \cos\theta & \\ & & & 1 \end{bmatrix}$$

- About z-axis
  - rotates $x \rightarrow y$

$$\begin{bmatrix} \cos\theta & -\sin\theta & & \\ \sin\theta & \cos\theta & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$$

- You may also specify rotation about an arbitrary axis.

#1 Current Matrix

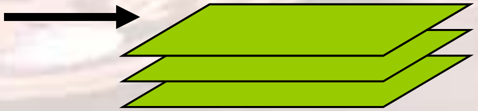
- An OpenGL matrix operation affects a **global 4x4 matrix.**
- It is the top matrix in the matrix stack you are currently working on. → **glMatrixMode**

Model View Matrix

glMatrixMode(GL_MODEL_VIEW)

glRotatef(1.0,0.0,0.0,1.0);

Current Matrix



M1=M1*R

Projection Matrix

glMatrixMode(GL_PROJECTION)

gluPerspective(…);

M2=M2*P

#1 Current Matrix

- When rendering, both of them are combined to transform the object.

Projection Matrix

Model View Matrix

MVP = (Projection)*(Model View)

V_Transform = MVP * V

#2 Last Transform Applied First

- OpenGL Post-multiply new transformation with current matrix when we call glRotate, glTranslate, or glScale.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \text{Current} \\ \text{ModelView} \\ \text{Matrix} \end{bmatrix} \times \begin{bmatrix} \text{glScale} \\ \text{Matrix} \end{bmatrix} = \begin{bmatrix} \text{New} \\ \text{ModelView} \\ \text{Matrix} \end{bmatrix}$$

- The last transformation is applied first to the object.

```
glLoadIdentity();

glRotatef(1.0,0.0,0.0,1.0);

glTranslatef(0.5,0.5,0.5);
```

M=IRT

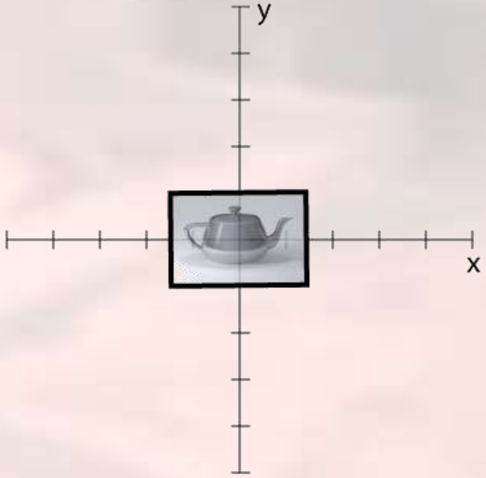
```
glLoadIdentity();

glTranslatef(0.5,0.5,0.5);

glRotatef(1.0,0.0,0.0,1.0);
```

M=ITR

Exercise

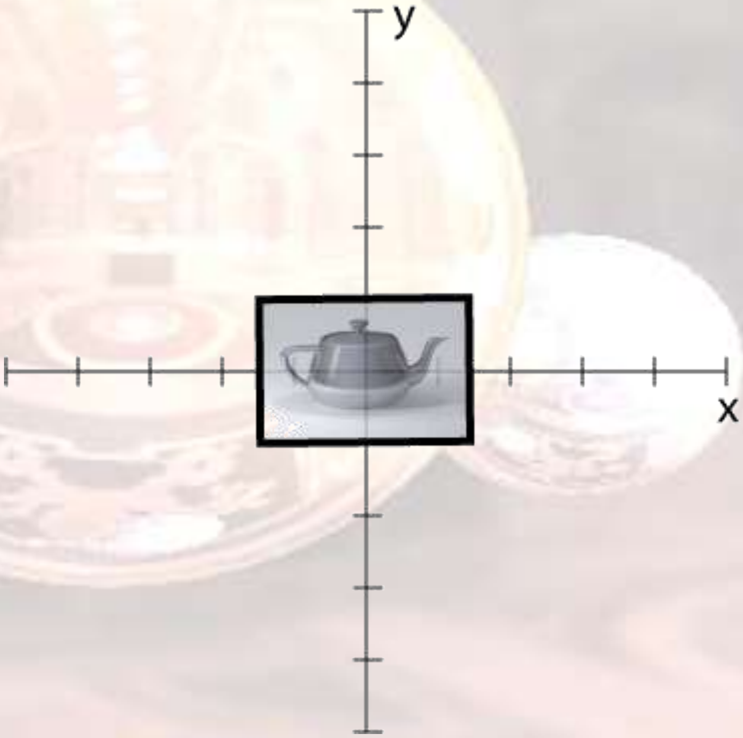
Draw the result of the following OpenGL transformation code.

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glScalef(1.5, 1.0, 1.0);
glRotatef(90.0, 0.0, 0.0, 1.0);
glTranslatef(2.0, 2.0, 0.0);
draw_teapot_image();
```



Exercise

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glScalef(1.5, 1.0, 1.0);
glRotatef(90.0, 0.0, 0.0, 1.0);
glTranslatef(2.0, 2.0, 0.0);
draw_teapot_image();
```



Exercise

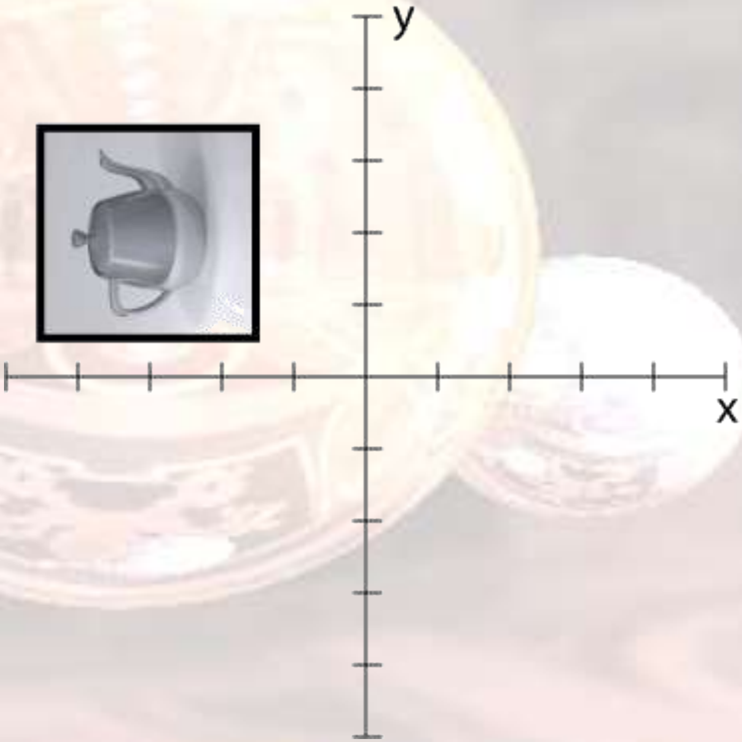
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glScalef(1.5, 1.0, 1.0);
glRotatef(90.0, 0.0, 0.0, 1.0);
glTranslatef(2.0, 2.0, 0.0);
draw_teapot_image();
```

Exercise

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glScalef(1.5, 1.0, 1.0);
glRotatef(90.0, 0.0, 0.0, 1.0);
glTranslatef(2.0, 2.0, 0.0);
draw_teapot_image();
```

Exercise

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glScalef(1.5, 1.0, 1.0);
glRotatef(90.0, 0.0, 0.0, 1.0);
glTranslatef(2.0, 2.0, 0.0);
draw_teapot_image();
```



Useful OpenGL Matrix Ops.

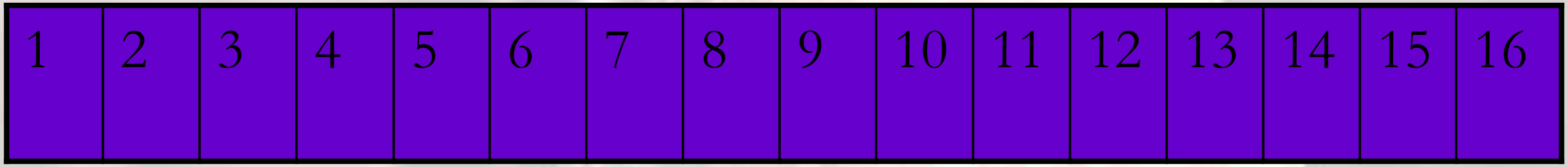
- glLoadIdentity : M = I
- glScale : M = MS
- glTranslate : M = MT
- glRotate : Specify rotation axis, angle. M = MR

Useful OpenGL Matrix Ops.

- glLoadMatrix(**Mo**) : M = Mo
- glGetFloat(**MatrixMode,Mo**) : Mo = M
- glMultMatrix(**Mo**) : M = M*Mo
- Caveat : OpenGL store matrix in **"Column Major"** instead of **"Row Major"**

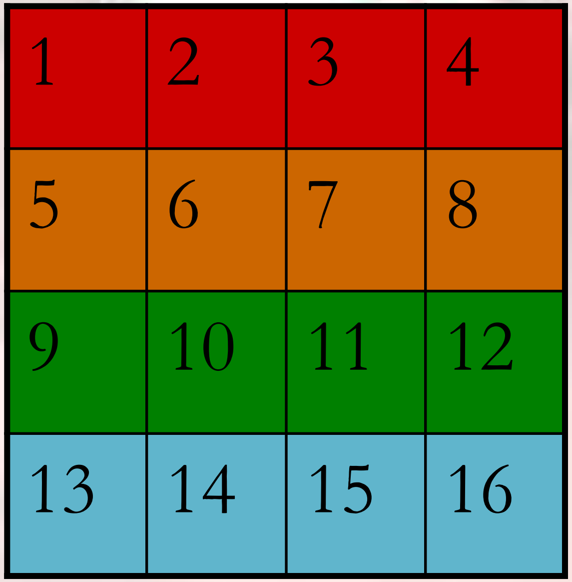
Column Major

Given a 1D array of 16 floats :



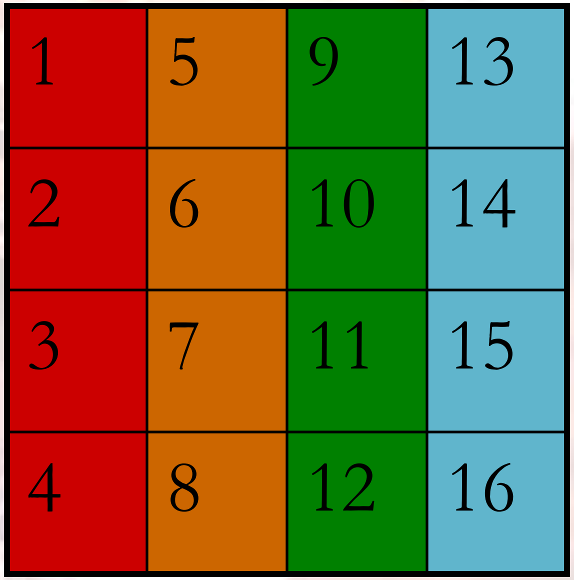
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

2D array in C :



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Matrix in OpenGL :



1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

Pre-multiply ?

- What to do if you want to pre-multiply the matrix ?

$M=RM$?

- Make use of “glGetFloat” & “glMultMatrix”.

```
glLoadIdentity();
glTranslatef(0.3,0.3,0.2);
glGetFloat(MODEL_VIEW,tempM);
glLoadIdentity();
glRotatef(1.0,0.0,0.0,1.0);
glMultMatrix(tempM);
```

$tempM=M=IT$

$M=IRtempM$

- Useful for updating transformation with UI control.

MP1 : Mesh Rendering

- Due on Sep. 25, 2012 at 3:30pm
  - Compass is sometimes not very stable. Try to submit earlier.
  - Email me if you encounter last minute failure on Compass.
- Depth Test : "**glEnable(GL_DEPTH_TEST);**"
- glRotate3f :
  - OpenGL will normalize the axis.

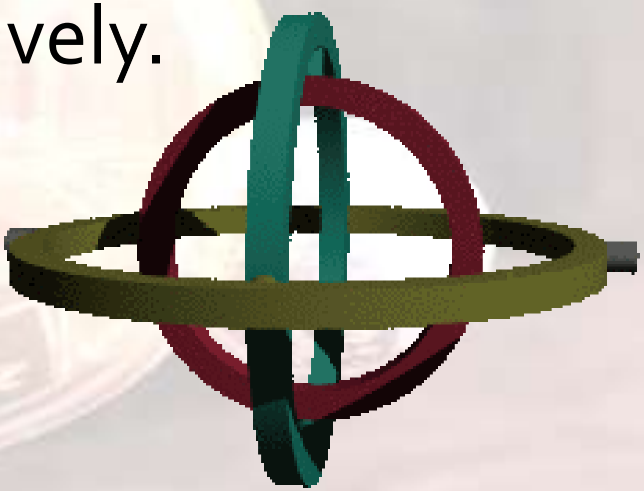
Interactive Viewing

- Interactive viewing is desired for 3D model display.
- Adjust the orientation of shape with UI
  - FPS style : Changing the first person view
    → Exploring the environment
  - ArcBall (TrackBall) : Rotate the object at view center.
    → Easier to view a single object in all direction

Euler Angles

- At most 75% of credit if you only implement Euler Angles.
- Rotate about X,Y,Z axis respectively.
- Very easy to implement.
- Keep track of X,Y,Z angles.

```
glRotatef(angleX,1,0,0);
glRotatef(angleY,0,1,0);
glRotatef(angleZ,0,0,1);
drawObject();
```

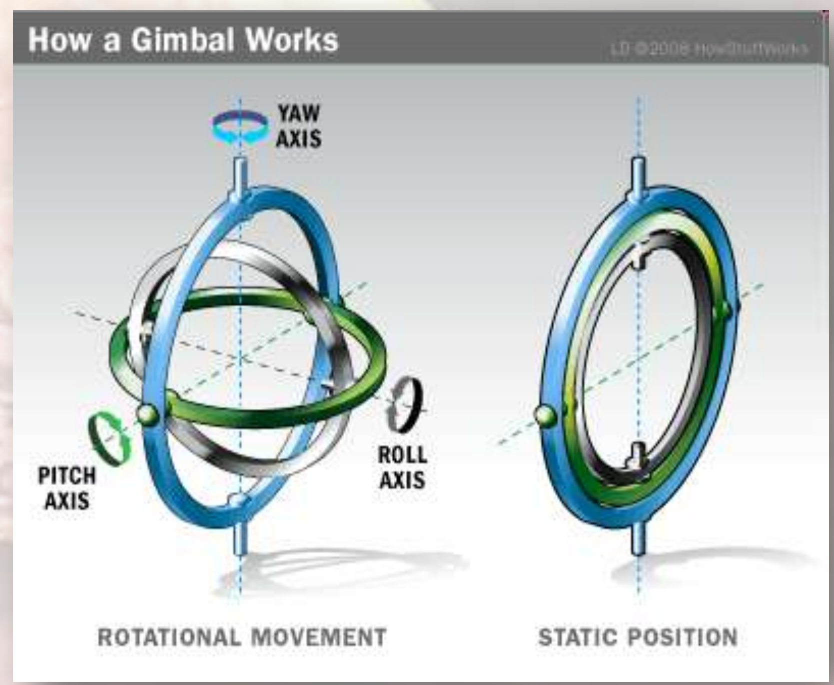


Gyroscope
(From Wikipedia)

```
gluUnProject(mouse_x, mouse_y, 0.0, modelview_matrix,
   projection_matrix, viewport_matrix, &x, &y, &z)
```

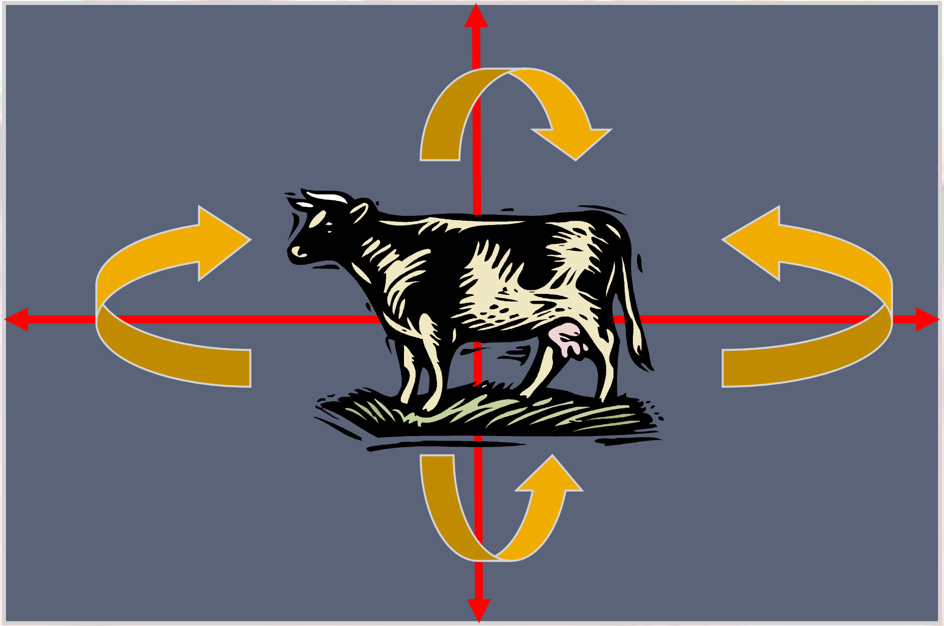
Euler Angles

- Problem : Gimbal Lock
- Occurs when two axes are aligned
- Second and third rotations have effect of transforming earlier rotations
    - ex: Rot x, Rot y, Rot z
        - If Rot y = 90 degrees, Rot z == -Rot x



Arcball Interface

- Intuition : Make use of the mouse position to control object orientation
  - Rotate object about some axis based on mouse movement



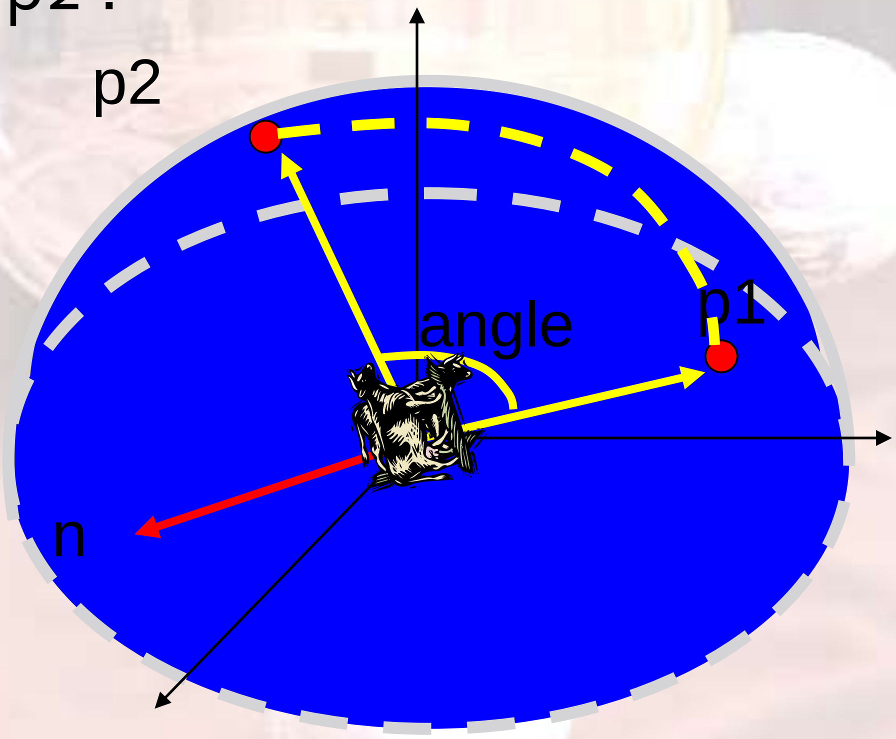
Arcball Interface

- Keep track a global rotation matrix **Rg**.
- Whenever there is a mouse movement, create a new rotation **Rn**.
- Update global rotation matrix **Rg** = **Rn*Rg**.

- How to define **Rn** ?

Arcball Interface

- To define a rotation : axis & angle
- Think of orientation as a point on the unit hemi-sphere
- How to rotate p1 to p2 ?



n = p1Xp2

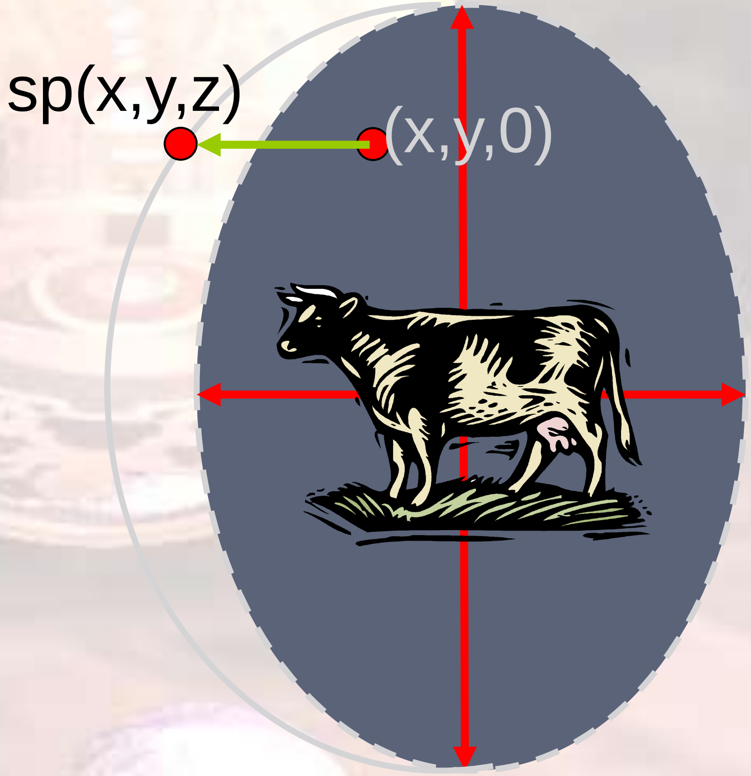
axis = n/|n|

|n| = sin(angle)

angle = asin(|n|)

Arcball Interface

- How to find a point on sphere based on normalized screen coordinates ?
- Map a 2D point (x,y) back to a unit sphere
  - z = sqrt(1 – x*x – y*y)



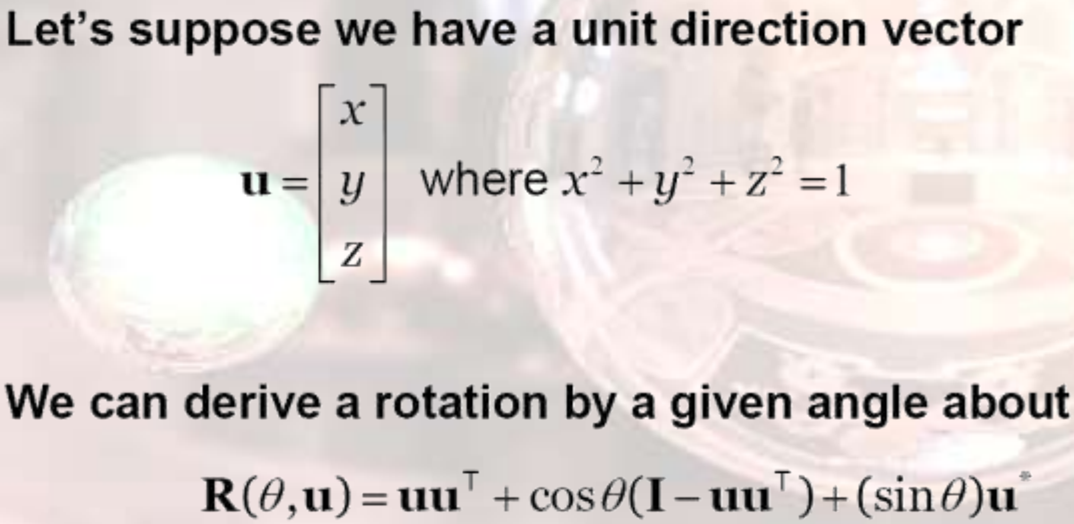
Arcball Interface

- Summary:
  - Get start/end mouse 2D position (glutMotion)
  - Map them to 3D points v1,v2 on hemi-sphere
  - Find rotation axis/angles from v1,v2

  - Update object orientation with rotation axis/angle
    - (Pre-multiply new rotation with current rotation)

Rotation About Any Axis

- Check lecture note :



Let's suppose we have a unit direction vector

$$\mathbf{u} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad \text{where } x^2 + y^2 + z^2 = 1$$

We can derive a rotation by a given angle about this axis

$$\mathbf{R}(\theta, \mathbf{u}) = \mathbf{u}\mathbf{u}^{\top} + \cos\theta(\mathbf{I} - \mathbf{u}\mathbf{u}^{\top}) + (\sin\theta)\mathbf{u}^{*}$$

- You can also call glRotate3f to generate it.

Rendering Accleration

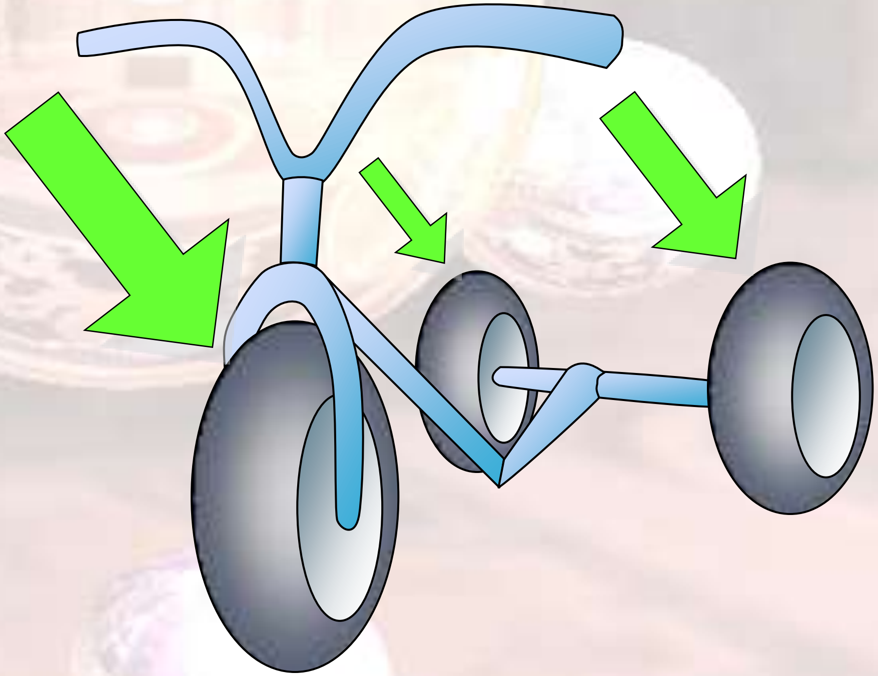
- Calling glBegin/glEnd is not optimal.
  - Many function calls
  - Repeated vertices
  - Data transfer
- Acceleration :
  - Method 1: Display List
  - Method 2: VertexArray
  - Method 3: Vertex Buffer Object (VBO)

Display Lists

Method One

Display Lists

A display list is a convenient and efficient way to name and organize a set of OpenGL commands.



glCallList(wheel_id);

modelview transformation

glCallList(wheel_id);

modelview transformation

glCallList(wheel_id);

Display Lists

To optimize performance, an OpenGL display list is a cache of commands rather than a dynamic database.

In other words, once a display list is created, it can't be modified on the fly.

Display List

- **A Display List is simply a group of OpenGL commands and arguments**

- **Most OpenGL drivers compile and accelerate Display Lists by**
  - storing all static data on video ram
  - optimizing OpenGL commands execution
  - Frustum & occlusion culling

- **Small driver overhead**
- **No time expensive data transfer**

Display List

- Usage : Create a new list
  - Call glBegin/glEnd /glVertex to store commands in the display list.
  - glCallList to reuse a display list.

glGenList

glNewList

glEndList

glCallList

…..

Red Book Sixth Edition : Chapter 7.

Vertex Arrays

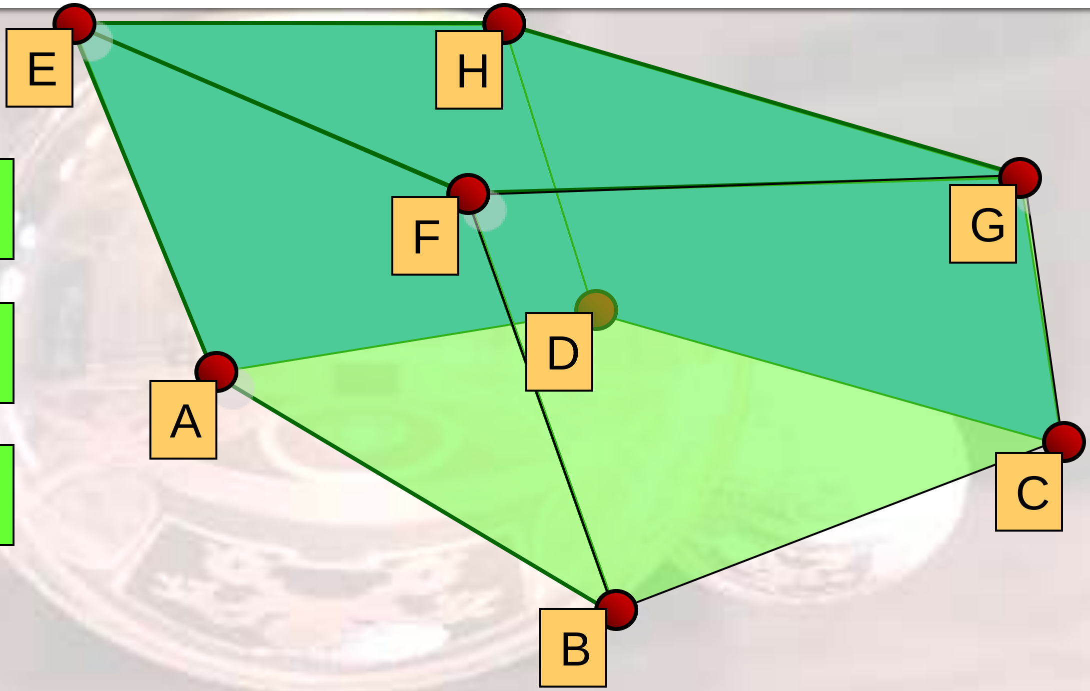
Method Two

The Basic Idea

Vertices Stored in an Array

0	0	0	0	0	1	1	0
1	1	0	0	0	1	0	0
1	1	1	1	1	1	1	0

Vertex G



Indices of Quads into the vertex array

A	B	C	D	A	B	F	E	B	C	G	F	E	F	G	H

A	D	H	E	D	C	G	H

Vertex Arrays

- **Similar to conventional approach, but: One driver call for all vertices**
  - small driver overhead
- **Data resides in CPU memory.**
  - Easier to update
- **Still transfering all vertices**
  - lot of transfer (CPU/AGP-bound bottleneck)

Vertex Arrays

- Usage : Enable client state for vertex array.
  - Provide pointers to your veritces/faces in memory.
  - Call glDrawElement to rendering everything at once.

glEnableClientState

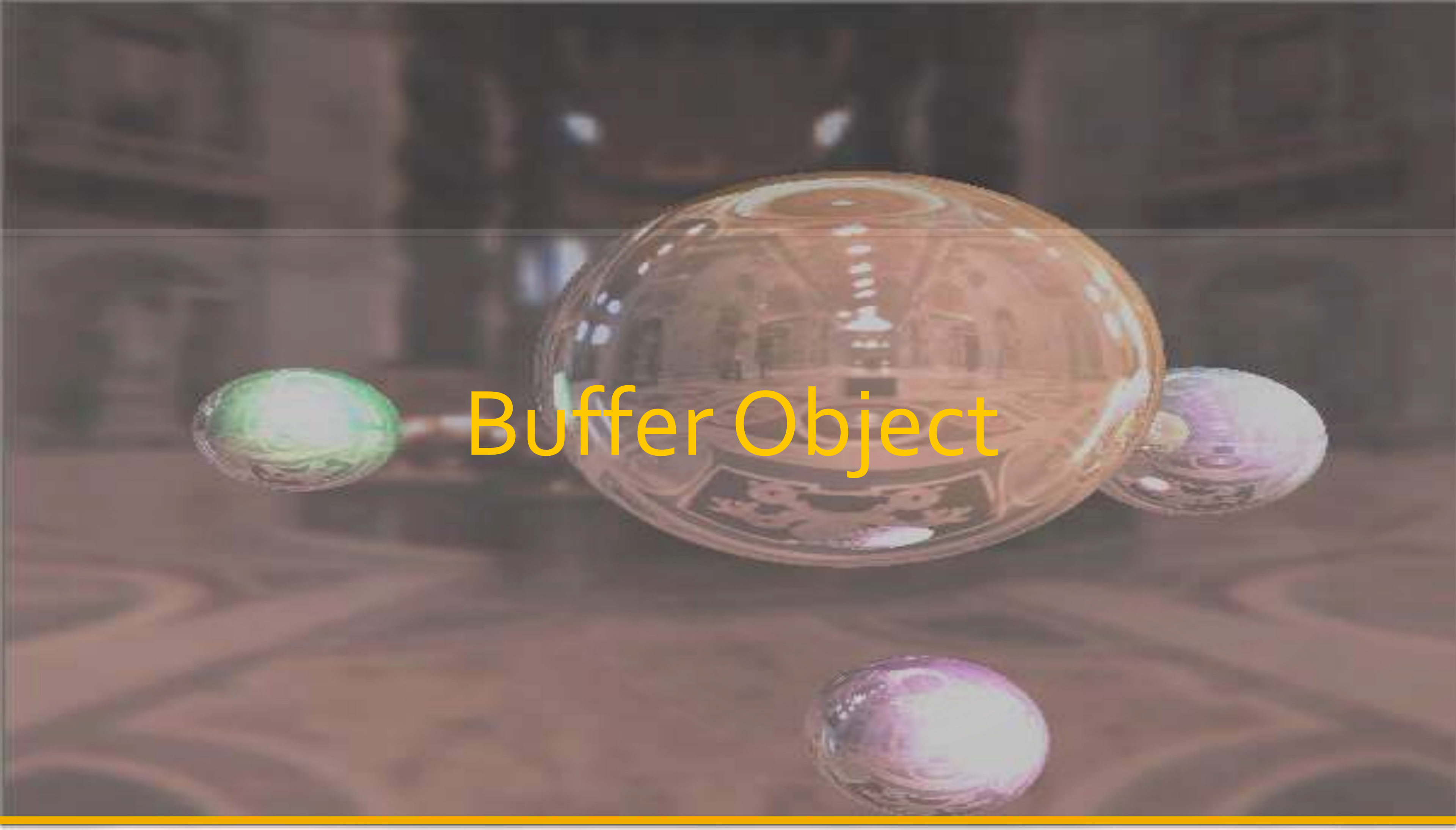
glVertexPointer

glColorPointer

glDrawElements

…..

Refer to Red Book for more information

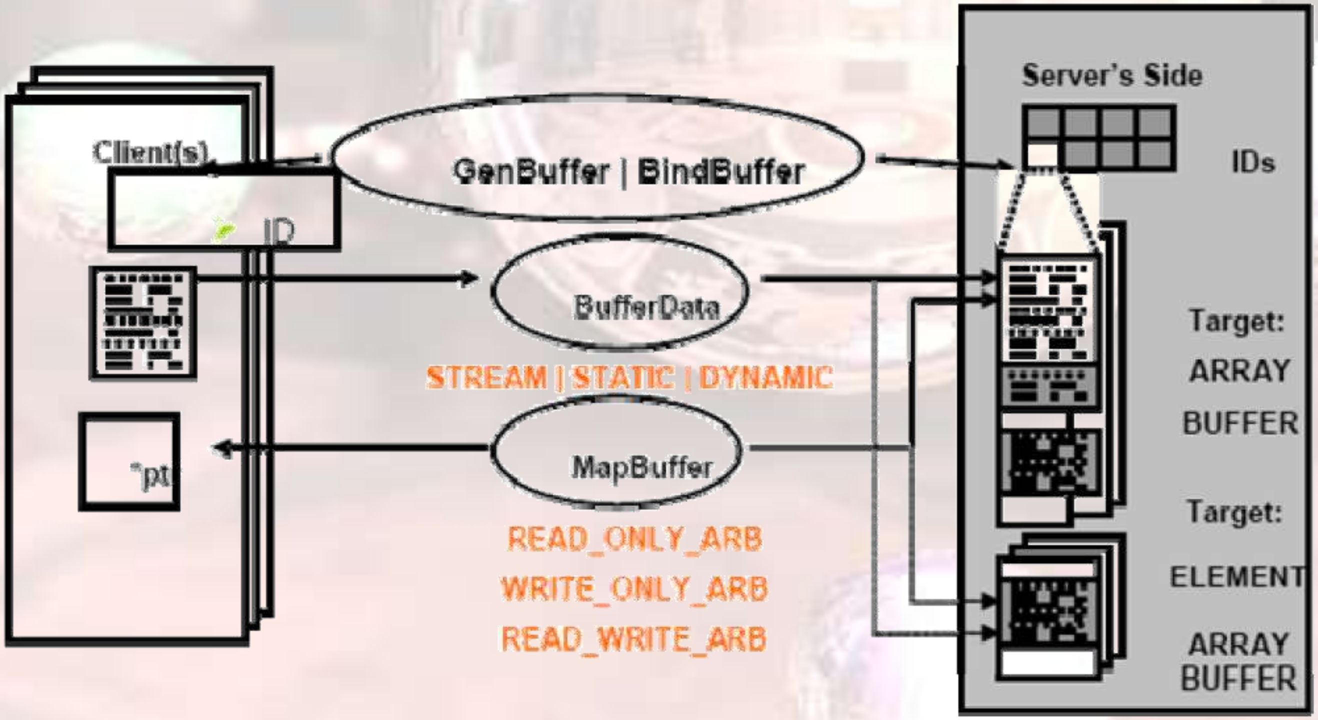


Buffer Object

Method Three

Vertex Buffer Object (VBO)

- **A vertex buffer object (VBO) is a powerful feature that allows storing vertex data in video ram**



Vertex Buffer Object (VBO)

- Very similar to vertex arrays
- VBOs hold geometry and state on the graphics hardware
  - Significant reduction in rendering time
- Provide mapping from application memory to graphics memory
  - Allows fast updates when geometry changes

Vertex Buffer Object

- Usage : Allocate enough buffer space in video memory.
  - Maps buffer memory to represent vertex/indices data.
  - Render as vertex arrays.

glGenBuffers

glBindBuffers

glBufferData

…..

Refer to the Red Book for more details

Summary

- Use Display Lists or Vertex Buffer Objects to store static objects
- Vertex Arrays or dynamic Vertex Buffer for deformable objects
- DrawElements is expensive
  - draw as many Triangles per DrawElements as possible
- Keep data transfer as small as possible

Q&A