Lecture 06

- View Control
- Modeling Transformation
- Simple Shadow
- Vertex-Buffer Object (VBO)
- 3D Version of Bouncing Ball

View Control

- Problem of directly adding/subtracting pitch, heading, and bank angles.
- Need a natural rotation of the view point.

One solution: Retain a view matrix $\mathbf{R}_{\mathbf{view}}$ instead of (h,p,b)

- For mouse movement dx,dy (normalized by the window size), the view matrix can be updated as:

  $$\mathbf{R}'_{\mathbf{view}} = \mathbf{R}_{\mathbf{Y}}\mathbf{R}_{\mathbf{X}}\mathbf{R}_{\mathbf{view}}$$

  where, $\mathbf{R}_{\mathbf{Y}}$ is a rotation about the Y axis by dx, and $\mathbf{R}_{\mathbf{X}}$ is a rotation about the X axis by dy.
- This method is good. But, with some problems:
  - It requires nine numbers as opposed to three numbers. When you want to save a view information, you need to save some redundant information.
  - Three column vectors of $\mathbf{R}_{\mathbf{view}}$ is supposed to be orthogonal to each other. However, these vectors become non-orthogonal after rotated many times due to numerical errors. Might create some rendering artifacts.

Another Solution: Solving Spherical Trigonometry?

- Heading, Pitch, and Bank are analogous to Longitude, Latitude, and Compass direction.
- Compass direction and Longitude become degenerate at north and south poles, so do heading and bank at pitch= ±90 degree

- One option: Solve spherical trigonometry https://en.wikipedia.org/wiki/Spherical_trigonometry

$$\cos a = \cos b \cos c + \sin b \sin c \cos A,$$
$$\cos b = \cos c \cos a + \sin c \sin a \cos B,$$
$$\cos c = \cos a \cos b + \sin a \sin b \cos C,$$
$$\frac{\sin A}{\sin a} = \frac{\sin B}{\sin b} = \frac{\sin C}{\sin c}.$$

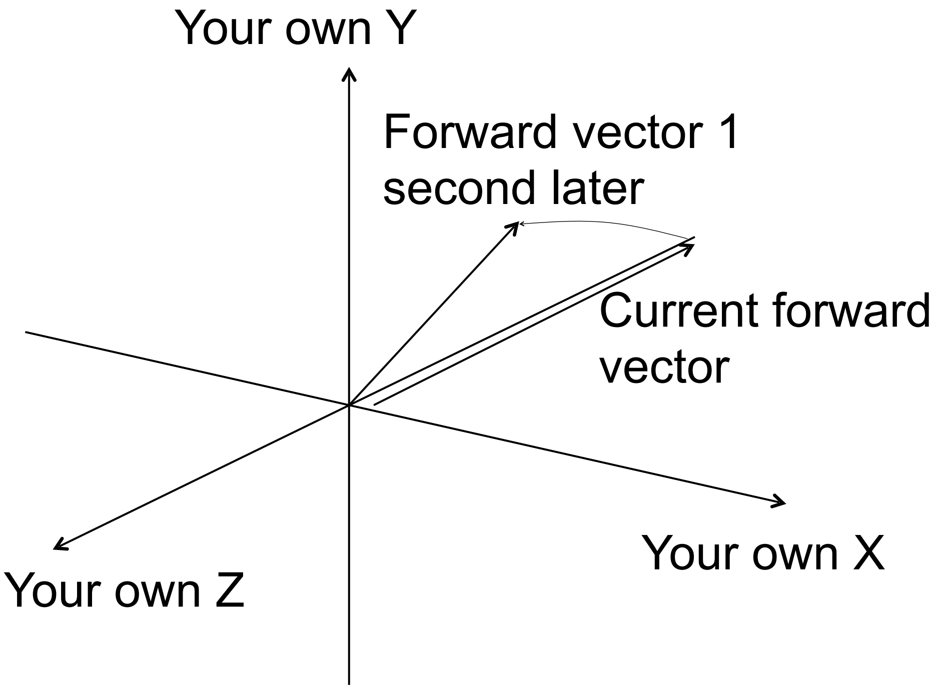
- You can try. I did. I ended up with getting zero divided by zero near the poles. The computation becomes very unstable.

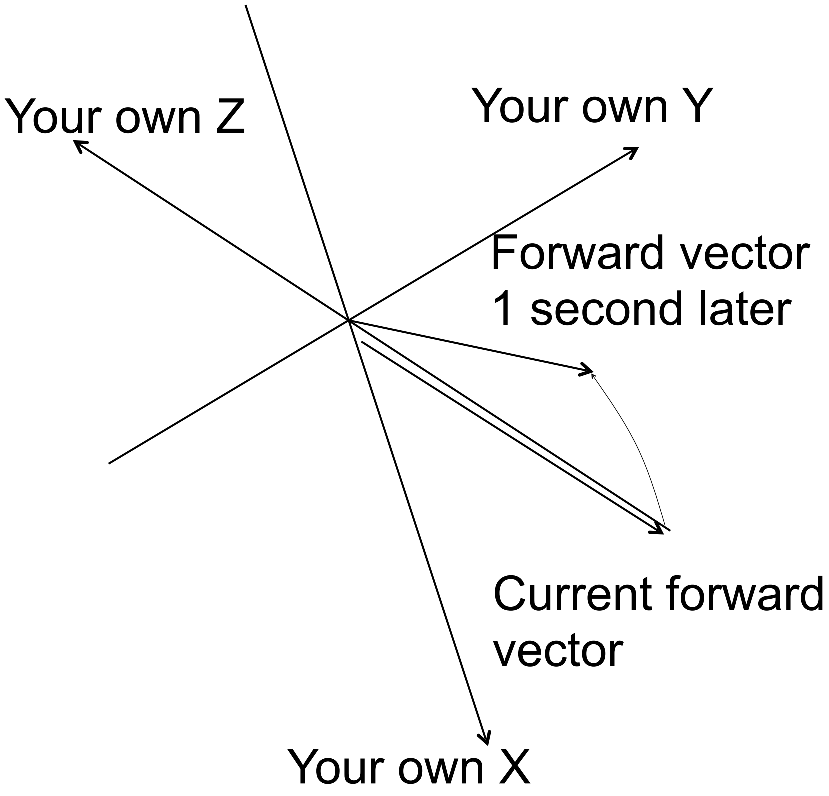
- Stable solution: Using forward- and up-vectors. (In fact, these are the third and second column vectors of the rotational matrix.)
  1. Calculate forward- and up-vectors of the next time step
  2. Calculate (h,p,b) from the next forward- and up-vectors.

Question:

- Let’s say you are rotating about your own Y axis 30 degrees per second. You are staying at the same location.
- Assume you can see yourself in the future.
- What does your 1-second future look like?

- Answer:
  - No matter what orientation you start with, you are looking 30 degrees off from the current position in your own coordinate system.





- For the same logic, if you are rotating about your X-axis, your future forward vector will be pointing upward in your own coordinate system.

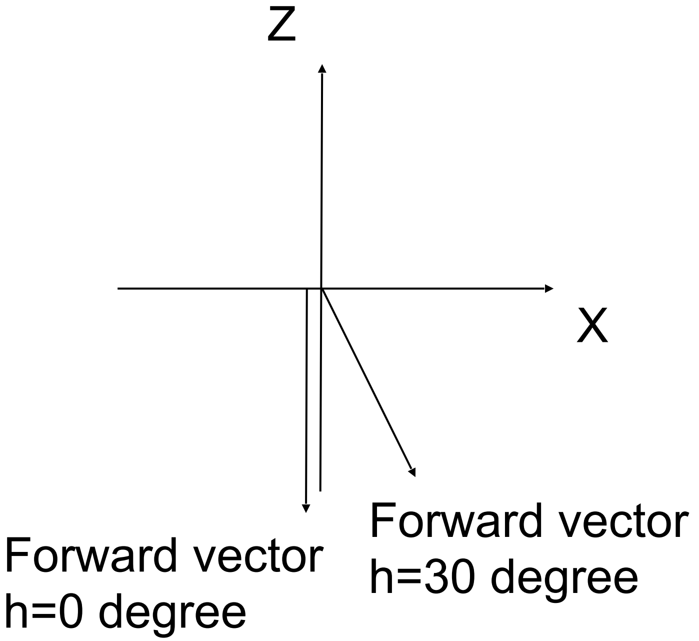
- When the user moves the mouse pointer horizontally, your orientation must rotate about your own Y axis.
- Not the about the Y-axis of the world coordinate system.

Next Question:

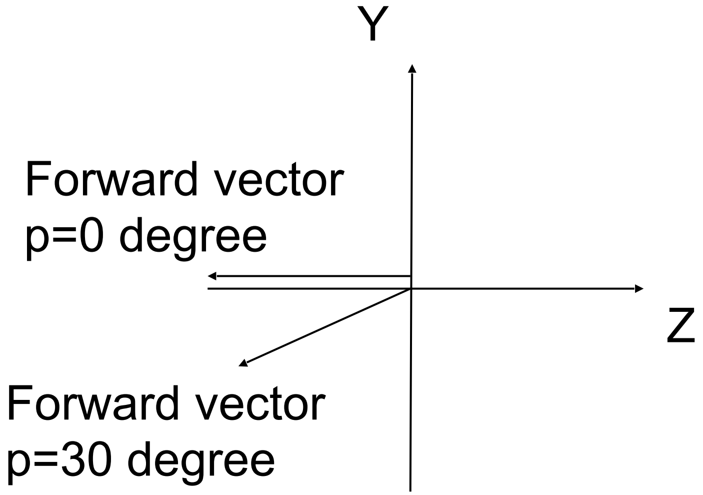
- Can I calculate (h,p,b) if I know the forward vector and the up vector?
- What about p=±90 degree?
  - Heading and bank angles are indistinctive. But, an attitude can be calculated for any set of heading and bank angles.
- In fact, heading and pitch can be calculated only from the forward vector.
- Up vector will give the bank angle.

- Calculating heading and pitch from the forward vector.
- Tentatively assume bank=0

Heading angle = atan2(fv.x,-fv.z)

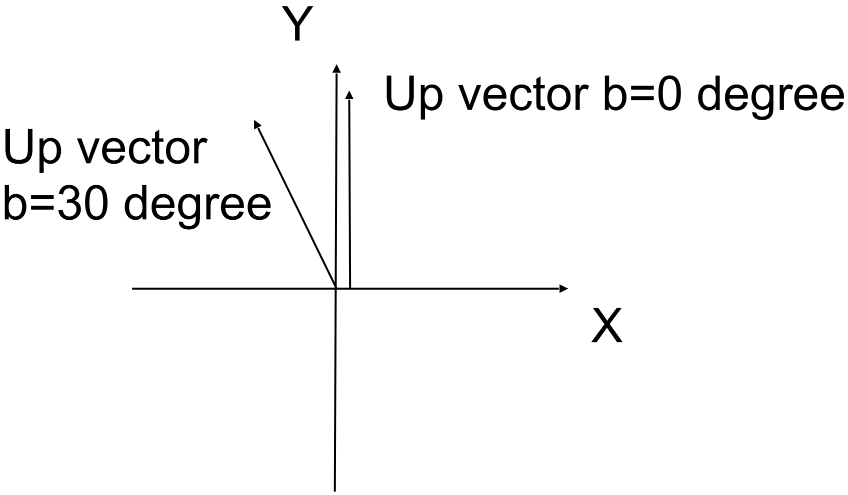


Pitch angle = asin(-fv.y)



- Transform up vector to the local coordinate by the tentative (h,p,b)
- Calculate actual bank angle by the transformed up vector.

Bank angle = atan2(uv.x,uv.y)



Y

Up vector b=0 degree

Up vector
b=30 degree

X

1. Create a new project from lighting example.
2. Change TARGET_NAME to glsl3d_viewcontrol
3. Add two member variables:

   ```
   int prevMx,prevMy;
   ```

4. Initialize prevMx,prevMy in the constructor. (Make them both zero).
5. Insert code for view-rotation in Interval function.

```
int wid,hei;
FsGetWindowSize(wid,hei);

int lb,mb,rb,mx,my;
auto evt=FsGetMouseEvent(lb,mb,rb,mx,my);
if(0!=lb && (mx!=prevMx || my!=prevMy))
{
    double denom=(double)YsGreater(wid,hei);
    double dx=2.0*(double)(prevMx-mx)/denom;
    double dy=2.0*(double)(prevMy-my)/denom;
```

Calculate mouse movement relative to the window size.

```
    YsVec3 fv(0,0,-1),uv(0,1,0);
    fv.RotateXZ(-dx);
    fv.RotateZY(-dy);
    uv.RotateXZ(-dx);
    uv.RotateZY(-dy);
```

Next forward and up vector in the current camera coordinate.

```
    YsMatrix3x3 view; // Camera to World
    view.RotateXZ(h);
    view.RotateZY(p);
    view.RotateXY(b);
    fv=view*fv;
    uv=view*uv;
```

Transforming next forward and up vectors into the world coordinate system.

```
    h=atan2(fv.x(),-fv.z());
    p=asin(YsBound(-fv.y(),-1.0,1.0));
    b=0.0; // Tentative
```

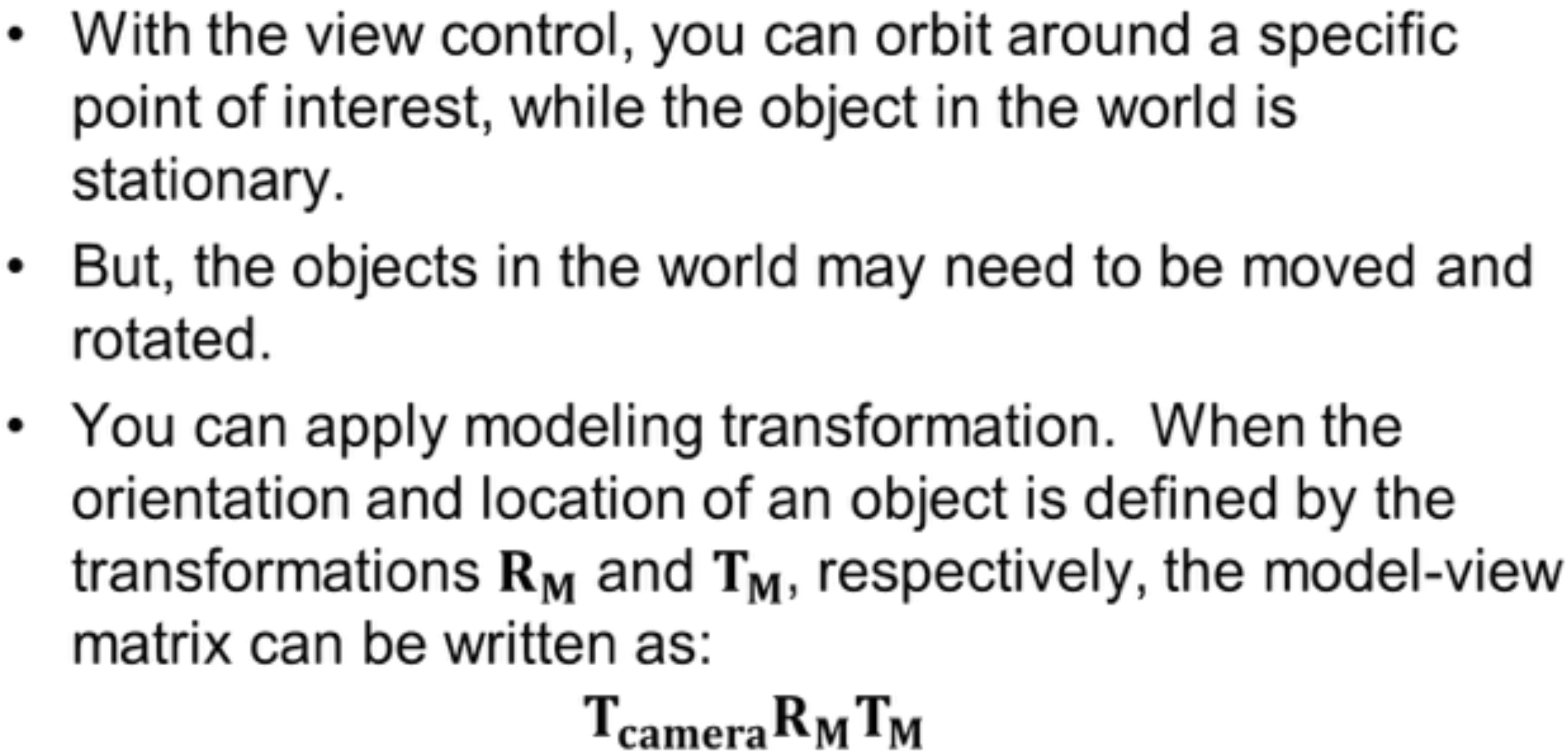
Calculate heading and pitch from the forward vector. Tentatively let bank=0.

```
    YsMatrix3x3 newView;
    newView.RotateXZ(h);
    newView.RotateZY(p);
    newView.RotateXY(b); // In fact it's no rotation.
    newView.MulInverse(uv,uv);
    b=atan2(-uv.x(),uv.y());
}

prevMx=mx;
prevMy=my;
```

Transform up vector to the tentative next camera orientation, and calculate actual bank angle.

Modeling transformation



- With the view control, you can orbit around a specific point of interest, while the object in the world is stationary.
- But, the objects in the world may need to be moved and rotated.
- You can apply modeling transformation. When the orientation and location of an object is defined by the transformations $\mathbf{R_M}$ and $\mathbf{T_M}$, respectively, the model-view matrix can be written as:

$$\mathbf{T_{camera} R_M T_M}$$

Let’s spin a cube.

1. Make a copy of the project from view-control example, and change TARGET_NAME to glsl3d_modeling_transformation
2. Add member variables:

   ```
   YsAtt3 cubeAtt;
   YsVec3 cubePos;
   ```

3. YsAtt3 is a class that stores an *attitude* by heading, pitch, and bank angles.
4. Initialize the member variables in constructor:

   ```
   cubeAtt.Set(0.0,0.0,0.0);
   cubePos.Set(0.0,0.0,0.0);
   ```

Attitude and Heading indicators of Cessna 172

5. Modify Draw function as:

```
YsMatrix4x4 view;
view.Translate(0,0,-viewDist);
view.RotateXY(-b);
view.RotateZY(-p);
view.RotateXZ(-h);
view.Translate(-viewTarget);

GLfloat viewMat[16];
view.GetOpenGlCompatibleMatrix(viewMat);
```

```
YsMatrix4x4 view;
view.Translate(0,0,-viewDist);
view.RotateXY(-b);
view.RotateZY(-p);
view.RotateXZ(-h);
view.Translate(-viewTarget);

YsMatrix4x4 modeling;
modeling.Translate(cubePos);
modeling.RotateXZ(cubeAtt.h());
modeling.RotateZY(cubeAtt.p());
modeling.RotateXY(cubeAtt.b());

YsMatrix4x4 fullMatrix=view*modeling;

GLfloat viewMat[16];
fullMatrix.GetOpenGlCompatibleMatrix(viewMat);
```

6. Add the following in Interval function.

```
auto a=(double)FsSubSecondTimer()/1000.0;
cubePos.Set(2.0*cos(a),0.0,2.0*sin(a));
cubeAtt.SetH(a);
```

Shadow

- An easy way to render a shadow is to draw everything with a dark color on a plane, such as y=0.
- It can be achieved by inserting a projection matrix between the view transformation and modeling transformation.

$$\mathbf{T_{camera}R_{M}P_{shadow}T_{M}}$$

- For example, if the shadow needs to be projected on y=-10,

$$\mathbf{P_{shadow}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -10 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- This does not solve an object casting a shadow on other objects. That can be done by a shadow-volume method or a shadow-mapping method, which I'll talk in a later lecture.

Cube with shadow

1. Make a copy of modeling-transformation example, and name it as glsl3d_simple_shadow.
2. Add a member variable in FsLazyWindowApplication:

   ```
   std::vector <GLfloat> cubeVtx;
   ```

3. Add a member function:

```
void FsLazyWindowApplication::MakeCube(double d)
{
    GLfloat df=(GLfloat)d;
    GLfloat vtxCol[]=
    {
        (Same contents as the previous example)
    };

    for(auto f : vtxCol)
    {
        cubeVtx.push_back(f);
    }
}
```

4. Modify Initialize function as:

```
void FsLazyWindowApplication::Initialize(int argc,char *argv[])
{
        YsGLSLRenderer::CreateSharedRenderer();
        MakeCube(5.0);
}
```

5. Make two versions of DrawPlainCube functions as:

```
void FsLazyWindowApplication::DrawPlainCube(
   YsGLSLShaded3DRenderer &renderer) const
{
        renderer.DrawVtxNomCol(GL_TRIANGLES,36,
       cubeVtx.data(),cubeVtx.data()+108,cubeVtx.data()+216);
}
void FsLazyWindowApplication::DrawPlainCube(
   YsGLSLPlain3DRenderer &renderer) const
{
        renderer.DrawVtx(GL_TRIANGLES,36,cubeVtx.data());
}
```

7. Insert following lines in Draw function:

```
YsMatrix4x4 shadowMat;
shadowMat.Translate(0.0,-12.0,0.0);
shadowMat.Scale(1.0,0.0,1.0);

fullMatrix=view*shadowMat*modeling;
fullMatrix.GetOpenGlCompatibleMatrix(viewMat);
{
    GLfloat color[]={0,0,0,1};

    YsGLSLPlain3DRenderer renderer;
    renderer.SetProjection(projMat);
    renderer.SetModelView(viewMat);
    renderer.SetUniformColor(color);
    DrawPlainCube(renderer);
}
```

Vertex-Buffer Object

- So, what happened to the display list?
- Display list is a feature of OpenGL 1.1.
- In the newer version OpenGL, you use Vertex-Buffer Object (VBO).
- You can pre-transfer arrays of vertex, normal, color, and all vertex attributes to the GPU’s memory space.
- It substantially reduces the CPU-GPU data transaction.

Using a VBO

Like a texture, you need an identifier of the VBO.

1. Reserve an identifier by glGenBuffers.
2. Bind the identifier to the GL_ARRAY_BUFFER by glBindBuffer.
3. Specify the size of the buffer by glBufferData
4. Transfer buffer data by glBufferSubData.

To use a buffer,

1. Bind the identifier to the GL_ARRAY_BUFFER.
2. Use data offset instead of the array pointer for the vertex-attribute buffers.
3. Draw primitives by glDrawArrays.
4. Unbind the identifier by glBindBuffer(GL_ARRAY_BUFFER, 0);

glsl3d_vertex_buffer_object

1. Copy simple-shadow project and make vbo project.
2. Write vertex_buffer_object.h – Since a vbo is a resource, it is a good idea to make a VertexBufferObject class to manage.
3. Include vertex_buffer_object.h

vertex_buffer_object.h

```
#ifndef VERTEX_BUFFER_OBJECT_IS_INCLUDED
#define VERTEX_BUFFER_OBJECT_IS_INCLUDED

#include <ysgl.h>

class VertexBufferObject
{
protected:
    GLuint vboIdent;

public:
    VertexBufferObject()
    {
        vboIdent=0;
    }
    ~VertexBufferObject()
    {
        CleanUp();
    }
    void CleanUp(void)
    {
        if(0!=vboIdent)
        {
            glDeleteBuffers(1,&vboIdent);
            vboIdent=0;
        }
    }
    void CreateBuffer(GLuint totalSizeInByte)
    {
        glGenBuffers(1,&vboIdent);
        glBindBuffer(GL_ARRAY_BUFFER,vboIdent);
        glBufferData(GL_ARRAY_BUFFER,
            totalSizeInByte,nullptr,GL_STATIC_DRAW);
    }
    GLuint GetVboIdent(void) const
    {
        return vboIdent;
    }
    GLuint GetZeroPointer(void)
    {
        return 0;
    }
    template <class T>
    GLuint PushBufferSubData(
        GLuint &currentPtr,GLuint n,const T incoming[])
    {
        auto returnPtr=currentPtr;
        auto bufLength=sizeof(T)*n;
        glBufferSubData(GL_ARRAY_BUFFER,
            currentPtr,bufLength,incoming);
        currentPtr+=bufLength;
        return returnPtr;
    }
};

class VertexBufferObjectVtxNomCol :
     public VertexBufferObject
{
public:
    GLuint vtxPtr,nomPtr,colPtr;

    VertexBufferObjectVtxNomCol()
    {
        vtxPtr=0;
        nomPtr=0;
        colPtr=0;
    }
};

#endif
```

4. Add member variable in FsLazyWindowApplication:

```
VertexBufferObjectVtxNomCol cubeVbo;
```

5. Remove cubeVtx member variable.
6. Modify MakeCube function as:

```
cubeVbo.CreateBuffer(sizeof(vtxNomCol));
auto currentPtr=cubeVbo.GetZeroPointer();
cubeVbo.vtxPtr=cubeVbo.PushBufferSubData(currentPtr,108,vtxNomCol);
cubeVbo.nomPtr=cubeVbo.PushBufferSubData(currentPtr,108,vtxNomCol+108);
cubeVbo.colPtr=cubeVbo.PushBufferSubData(currentPtr,144,vtxNomCol+216);
```

7. Remove loop for populating cubeVtx.

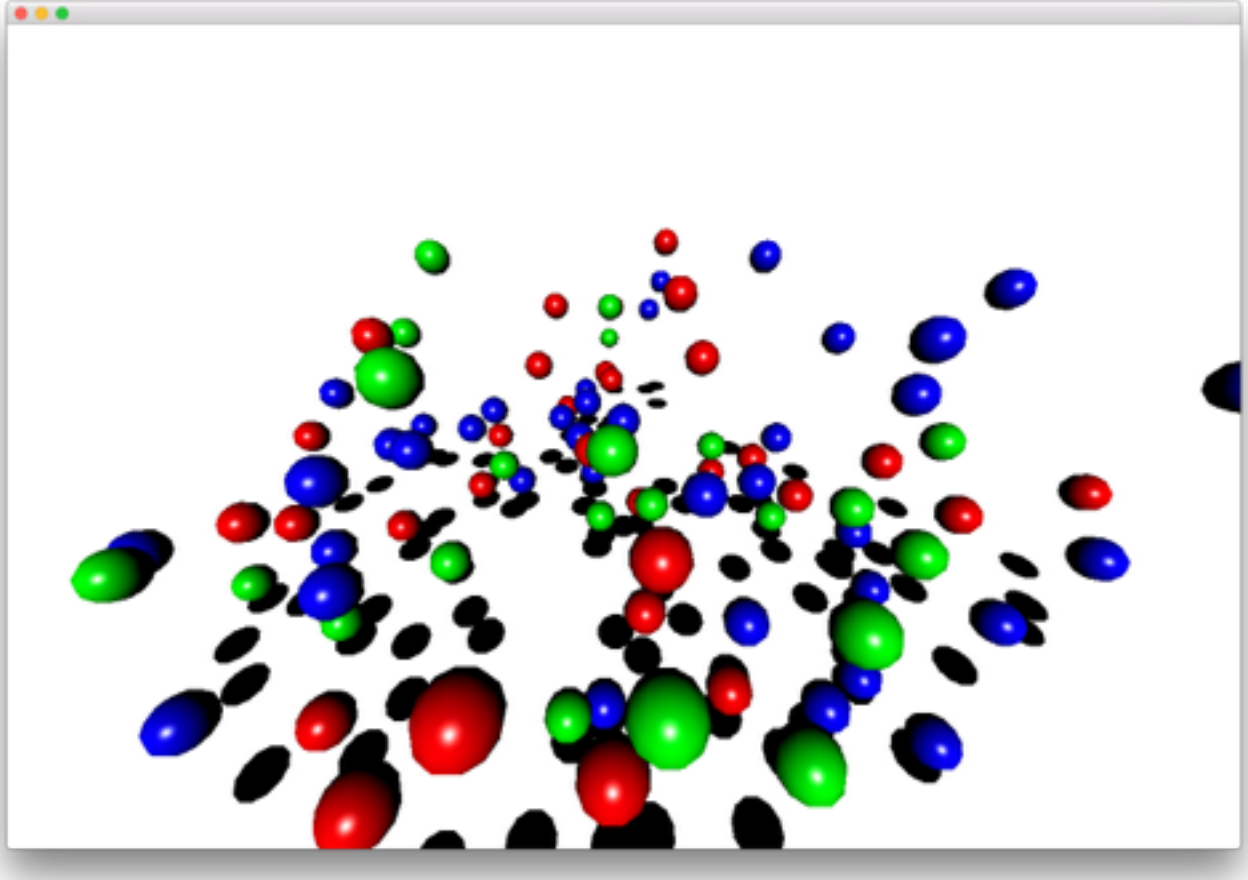
8. Modify DrawPlainCube functions as:

```
void FsLazyWindowApplication::DrawPlainCube(YsGLSLShaded3DRenderer &renderer) const
{
   glBindBuffer(GL_ARRAY_BUFFER,cubeVbo.GetVboIdent());
   renderer.DrawVtxNomCol(GL_TRIANGLES,
      36,(GLfloat *)cubeVbo.vtxPtr,(GLfloat *)cubeVbo.nomPtr,(GLfloat *)cubeVbo.colPtr);
   glBindBuffer(GL_ARRAY_BUFFER,0);
}
void FsLazyWindowApplication::DrawPlainCube(YsGLSLPlain3DRenderer &renderer) const
{
   glBindBuffer(GL_ARRAY_BUFFER,cubeVbo.GetVboIdent());
   renderer.DrawVtx(GL_TRIANGLES,36,(GLfloat *)cubeVbo.vtxPtr);
   glBindBuffer(GL_ARRAY_BUFFER,0);
}
```

- .

3D Bouncing Ball

- With VBO, the program can handle large count of 3D objects with small GPU-CPU transaction per frame.
- Let's make a 3D version of bouncing ball.



1. Copy vbo project and make bounce3d project.
2. Rename TARGET_NAME as glsl3d_bounce3d
3. Add sphereutil.h

sphereutli.h

```
#ifndef SPHEREUTIL_IS_INCLUDED
#define SPHEREUTIL_IS_INCLUDED

template <class T>
std::vector <T> MakeSphere(int nDiv)
{
    const double YsPi=3.14159265358979323;

    std::vector <T> vtx;
    const int nDivY=nDiv/2;
    const int nDivX=nDiv;

    for(int ip=0; ip<nDivY; ++ip)
    {
        const double p0=-YsPi/2.0+YsPi*(double)ip/(double)(nDivY);
        const double p1=-YsPi/2.0+YsPi*(double)(ip+1)/(double)(nDivY);

        const double y0=sin(p0);
        const double y1=sin(p1);
        const double lateral0=cos(p0);
        const double lateral1=cos(p1);
        for(int ih=0; ih<nDivX; ++ih)
        {
            const double h0=YsPi*2.0*(double)ih/(double)nDivX;
            const double h1=YsPi*2.0*(double)(ih+1)/(double)nDivX;

            const double x0=lateral0*cos(h0);
            const double z0=lateral0*sin(h0);
            const double x1=lateral0*cos(h1);
            const double z1=lateral0*sin(h1);
            const double x2=lateral1*cos(h0);
            const double z2=lateral1*sin(h0);
            const double x3=lateral1*cos(h1);
            const double z3=lateral1*sin(h1);

            const double tri[3*6]=
            {
                x0,y0,z0, x2,y1,z2, x3,y1,z3,
                x3,y1,z3, x1,y0,z1, x0,y0,z0
            };

            if(ip<nDivY-1)
            {
                vtx.push_back((T)tri[0]);
                vtx.push_back((T)tri[1]);
                vtx.push_back((T)tri[2]);
                vtx.push_back((T)tri[3]);
                vtx.push_back((T)tri[4]);
                vtx.push_back((T)tri[5]);
                vtx.push_back((T)tri[6]);
                vtx.push_back((T)tri[7]);
                vtx.push_back((T)tri[8]);
            }
            if(0<ip)
            {
                vtx.push_back((T)tri[ 9]);
                vtx.push_back((T)tri[10]);
                vtx.push_back((T)tri[11]);
                vtx.push_back((T)tri[12]);
                vtx.push_back((T)tri[13]);
                vtx.push_back((T)tri[14]);
                vtx.push_back((T)tri[15]);
                vtx.push_back((T)tri[16]);
                vtx.push_back((T)tri[17]);
            }
        }
    }
    return vtx;
}

#endif
```

4. Add Ball class as:

```
class Ball
{
public:
   GLfloat col[4];
   YsVec3 pos,vel;
};
```

5. Add enum and member variable in FsLazyWindowApplication as:

```
      enum
   {
      NUM_BALL=100
   };
   Ball ball[NUM_BALL];
   int nBallVtx;
```

6. Remove:

```
   YsAtt3 cubeAtt;
   YsVec3 cubePos;
```

7. Change cubeVbo -> ballVbo, MakeCube -> MakeBall
8. MakeBall function:

```
void FsLazyWindowApplication::MakeBall(void)
{
    auto sphereVtx=MakeSphere<GLfloat>(12);

    ballVbo.CreateBuffer(2*sizeof(GLfloat)*sphereVtx.size());
    auto currentPtr=ballVbo.GetZeroPointer();
    ballVbo.vtxPtr=ballVbo.PushBufferSubData(
       currentPtr,sphereVtx.size(),sphereVtx.data());
    ballVbo.nomPtr=ballVbo.PushBufferSubData(
        currentPtr,sphereVtx.size(),sphereVtx.data());
    nBallVtx=sphereVtx.size()/3;
}
```

9. DrawPlainCube -> DrawPlainBall (cubeVtx -> ballVtx, 36->nBallVtx);

10. In Interval function: Remove cube rotation.
11. View target is (0,10,0)
    (Balls bounce within (-20,-20,-20)-(20,20,20))
12. Add SetInitialLocationAndVelocity function and call from Initialize.

```
void FsLazyWindowApplication::SetInitialLocationAndVelocity(void)
{
    for(auto &b : ball)
    {
        int x=rand()%21-10;
        int y=10+rand()%21-10;
        int z=rand()%21-10

        int vx=rand()%21-10;
        int vy=rand()%21-10;
        int vz=rand()%21-10

        b.pos.Set(x,y,z);
        b.vel.Set(vx,vy,vz);
    }
}
```

13. Add the following three functions:
    - FsLazyWindowApplication::Move(const double dt);
    - Ball::Move(const double dt);
    - Ball::BounceOnWall(const YsVec3 &o,const YsVec3 &n)
14. Modify Draw function so that it draws all balls.

```
class Ball
{
public:
    GLfloat col[4];
    YsVec3 pos,vel;

    void Move(const double dt)
    {
        const double G=9.8;
        pos+=vel*dt;

        YsVec3 a(0.0,-G,0.0);
        vel+=a*dt;
    }
    void BounceOnWall(const YsVec3 &o,const YsVec3 &n)
    {
        const double radius=1.0;
        const double dist=(pos-o)*n;
        if(dist<radius && vel*n<0.0)
        {
            vel-=2.0*n*(vel*n);
        }
    }
};
```

```
void FsLazyWindowApplication::Move(const double dt)
{
    for(auto &b : ball)
    {
        b.Move(dt);
        b.BounceOnWall(YsVec3(0.0,0.0,0.0),YsYVec());
        b.BounceOnWall(YsVec3( 20.0,0.0,0.0),YsVec3(-1.0,0.0,0.0));
        b.BounceOnWall(YsVec3(-20.0,0.0,0.0),YsVec3( 1.0,0.0,0.0));
        b.BounceOnWall(YsVec3(0.0, 20.0,0.0),YsVec3(0.0,-1.0,0.0));
        b.BounceOnWall(YsVec3(0.0,-20.0,0.0),YsVec3(0.0, 1.0,0.0));
        b.BounceOnWall(YsVec3(0.0,0.0, 20.0),YsVec3(0.0,0.0,-1.0));
        b.BounceOnWall(YsVec3(0.0,0.0,-20.0),YsVec3(0.0,0.0, 1.0));
    }
}
```

Drawing all balls

```
for(auto &b : ball)
{
    YsMatrix4x4 translation;
    translation.Translate(b.pos);

    YsMatrix4x4 fullMatrix=view*translation;

    GLfloat viewMat[16];
    fullMatrix.GetOpenGlCompatibleMatrix(viewMat);

    {
        GLfloat lightDir[]={0,0,1};

        YsGLSLShaded3DRenderer renderer;
        renderer.SetProjection(projMat);
        renderer.SetModelView(viewMat);
        renderer.SetLightDirectionInCameraCoordinate(0,lightDir);
        renderer.SetUniformColor(b.col);
        DrawPlainBall(renderer);
    }

    YsMatrix4x4 shadowMat;
    shadowMat.Scale(1.0,0.0,1.0);

    fullMatrix=view*shadowMat*translation;
    fullMatrix.GetOpenGlCompatibleMatrix(viewMat);
    {
        GLfloat color[]={0,0,0,1};

        YsGLSLPlain3DRenderer renderer;
        renderer.SetProjection(projMat);
        renderer.SetModelView(viewMat);
        renderer.SetUniformColor(color);
        DrawPlainBall(renderer);
    }
}
```