Lecture 08

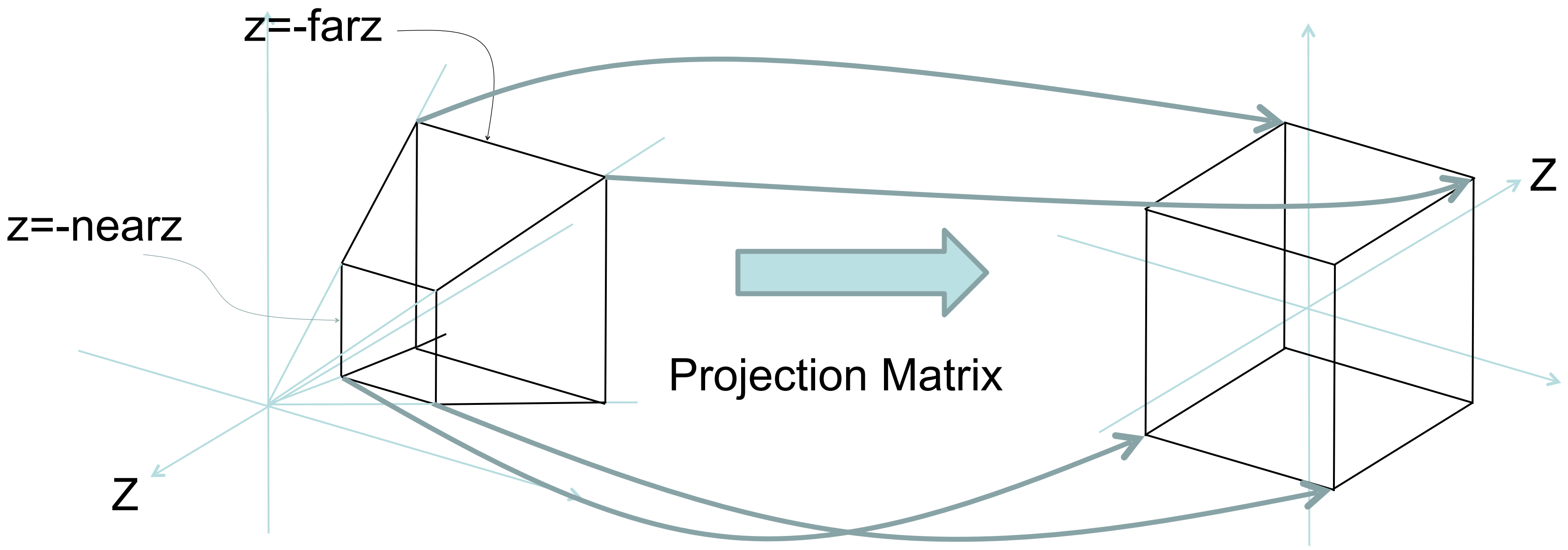
- Picking
- Bouncing Ball 3D
- Accelerate collision detection by 3D lattice
- Re-constructing connection of an STL model.

Picking

- Identifying which primitive is under the mouse cursor.
- Possible options:
  1. Using OpenGL's picking feature – Very poorly designed. May not be supported in the newer versions.
  2. Drawing primitives in different color, and then check the color of the pixel under the mouse cursor – The actual RGB value written to the pixel may be reduced to as low as 12 bits. The identification information may be lost.
  3. Transform mouse pointer to a 3D line by the inverse of projection and view matrix, and calculate intersection with the primitives. - The most reliable. Can easily be parallelized.

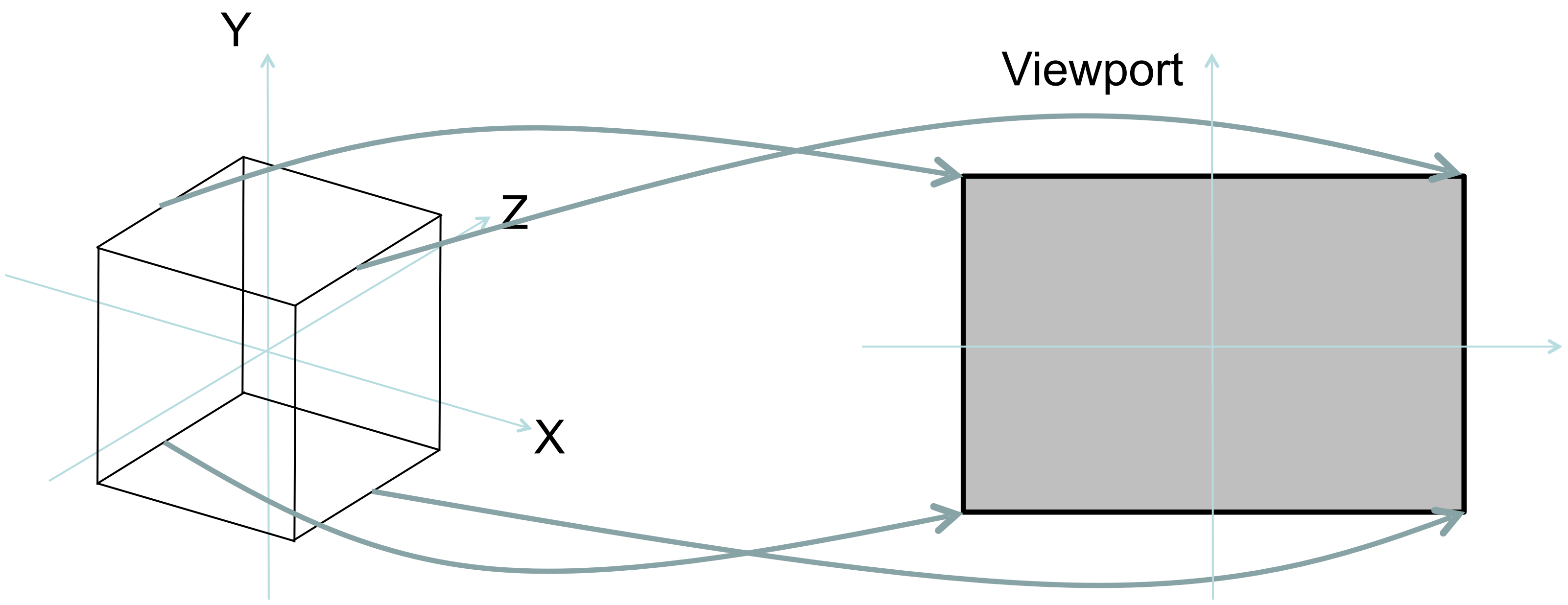
Projection transformation

- OpenGL projection transforms a view frustum into a cube (-1,-1,-1)-(1,1,1)
- Z direction will be inverted after the projection. Larger z means forward after the projection. (As shown in the thick arrow in the figure.)

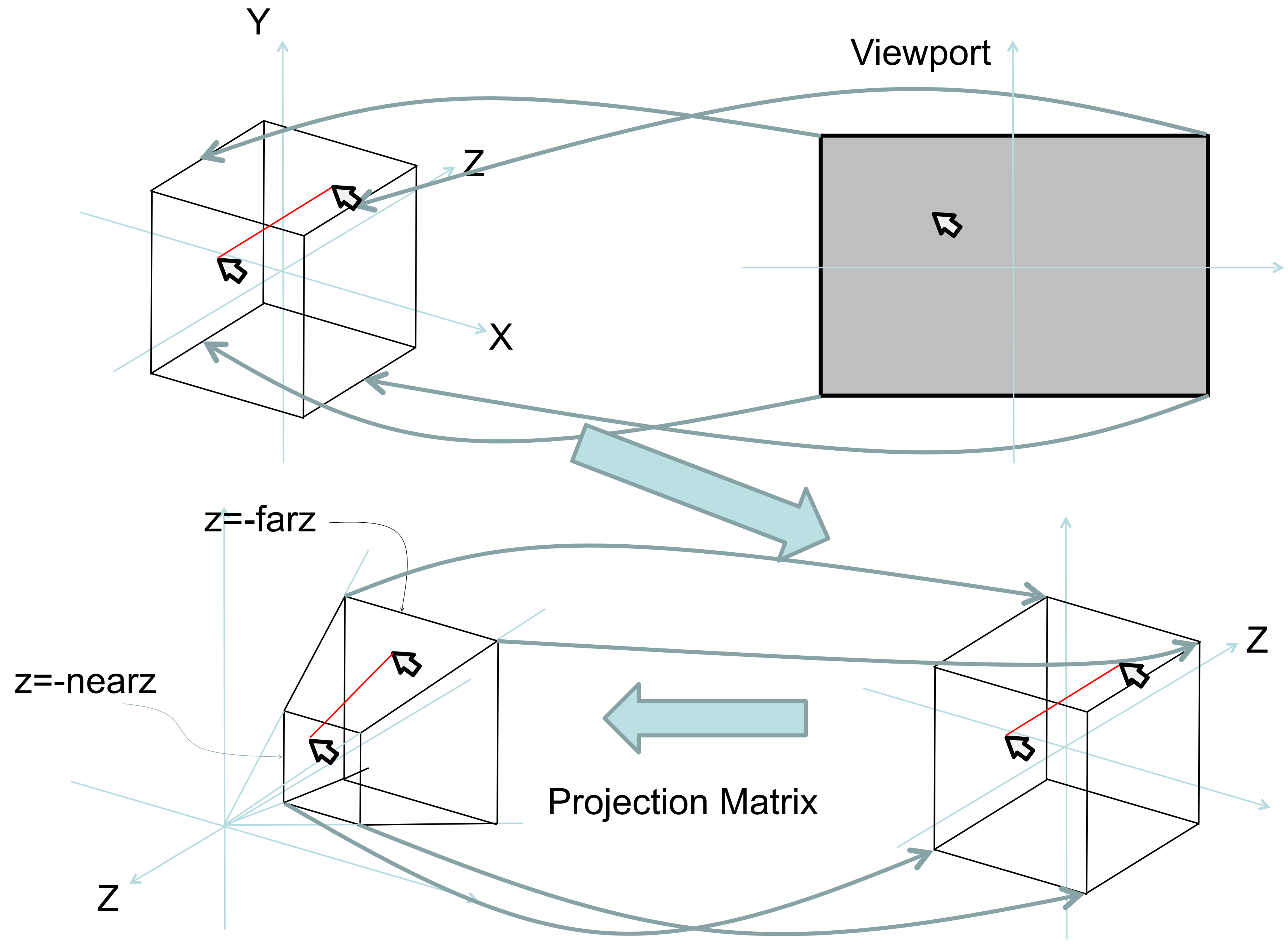


Viewport transformation

- (x,y)=(-1,-1) to (1,1) in the projected coordinate are linearly mapped to the viewport, which is in general same size as the window, specified by glViewport.



Mouse coordinate can be inverse-transformed to a line.



Transforming a mouse coordinate to a 3D line

1. Normalize window coordinate (x_s,y_s) to (-1,-1)-(1,1) with respect to the viewport => (x_v,y_v)
2. Inverse-transform $(x_v,y_v,-1)$ and $(x_v,y_v,1)$ by the projection matrix => (x_n,y_n,z_n), (x_f,y_f,z_f)
3. Inverse-transform (x_n,y_n,z_n) and (x_f,y_f,z_f) by the model-view matrix => (x_1,y_1,z_1), (x_2,y_2,z_2)

Let’s draw it on the screen.

1. Copy project from binary-stl sample, rename TARGET_NAME to glsl3d_mouse_to_line
2. Add member variables:
   ```
   YsVec3 lastClick[2];
   ```
3. Initialize lastClick[0] and lastClick[1] in the constructor as:
   ```
   lastClick[0]=YsVec3::Origin();
   lastClick[1]=YsVec3::Origin();
   ```

4. In Interval function, add:

```
if(evt==FSMOUSEEVENT_LBUTTONDOWN)
{
   double x=(double)mx/(double)wid;
   double y=(double)(hei-my)/(double)hei;
   x=x*2.0-1.0;
   y=y*2.0-1.0;

   lastClick[0].Set(x,y,-1.0);
   lastClick[1].Set(x,y, 1.0);
   for(auto &p : lastClick)
   {
      drawEnv.GetProjectionMatrix().MulInverse(p,p,1.0);
      drawEnv.GetViewMatrix().MulInverse(p,p,1.0);
   }
}
```

5. In Draw function,

```
{
   YsGLSLPlain3DRenderer renderer;  // Again, do not nest the renderer!
   renderer.SetProjection(projMat);
   renderer.SetModelView(viewMat);

   GLfloat color[8]={0,0,1,1, 1,0,0,1};
   const GLfloat vtx[6]=
   {
      lastClick[0].xf(),lastClick[0].yf(),lastClick[0].zf(),
      lastClick[1].xf(),lastClick[1].yf(),lastClick[1].zf()
   };
   renderer.DrawVtxCol(GL_LINES,2,vtx,color);
}
```

Picking

- Change color of the picked polygon.
- What needs to be done when the user clicks on the left button:
  1. Calculate a line from the mouse coordinate.
  2. Find the polygon that:
     - intersects with the line, and
     - in front of the camera, and
     - nearest to the view point.
  3. Change the color of the polygon.

Intersection of a line and a polygon

- Plane-line intersection.
- The intersecting point is on the boundary or inside of the polygon. (YsCheckInsidePolygon3)

1. Copy project from mouse-to-line, rename TARGET_NAME as glsl3d_picking
2. Add a function:

```
void FsLazyWindowApplication::MouseCoordinateTo3DLine(YsVec3 ln[2],int mx,int my) const
{
   int wid,hei;
   FsGetWindowSize(wid,hei);

   double x=(double)mx/(double)wid;
   double y=(double)(hei-my)/(double)hei;
   x=x*2.0-1.0;
   y=y*2.0-1.0;

   ln[0].Set(x,y,-1.0);
   ln[1].Set(x,y, 1.0);
   for(int i=0; i<2; ++i)
   {
      auto &p=ln[i];
      drawEnv.GetProjectionMatrix().MulInverse(p,p,1.0);
      drawEnv.GetViewMatrix().MulInverse(p,p,1.0);
   }
}
```

3. Replace:

```
if(evt==FSMOUSEEVENT_LBUTTONDOWN)
{
   double x=(double)mx/(double)wid;
   double y=(double)(hei-my)/(double)hei;
   x=x*2.0-1.0;
   y=y*2.0-1.0;

   lastClick[0].Set(x,y,-1.0);
   lastClick[1].Set(x,y, 1.0);
   for(auto &p : lastClick)
   {
      drawEnv.GetProjectionMatrix().MulInverse(p,p,1.0);
      drawEnv.GetViewMatrix().MulInverse(p,p,1.0);
   }
}
```

```
if(evt==FSMOUSEEVENT_LBUTTONDOWN)
{
   MouseCoordinateTo3DLine(lastClick,mx,my);
}
```

4. Add member variable:

```
std::vector <float> col;
```

5. In LoadBinaryStl, after calling ::LoadBinaryStl,

```
col.clear();
for(int i=0; i<vtx.size()/3; ++i)
{
    col.push_back(0);
    col.push_back(0);
    col.push_back(1);
    col.push_back(1);
}
```

6. In Draw function, replace:

```
GLfloat color[4]={0,0,1,1};
renderer.SetUniformColor(color);
renderer.DrawVtxNom(GL_TRIANGLES,vtx.size()/3,vtx.data(),nom.data());
```

⇩

```
renderer.DrawVtxNomCol(GL_TRIANGLES,vtx.size()/3,vtx.data(),nom.data(),col.data());
```

7. Add a function:

```
int FsLazyWindowApplication::PickedTriangle(int mx,int my) const
{
    YsVec3 ln[2];
    MouseCoordinateTo3DLine(ln,mx,my);

    const YsVec3 o=ln[0];
    const YsVec3 v=YsUnitVector(ln[1]-ln[0]);

    int picked=-1;
    double pickedDist=0.0;
    for(int i=0; i<vtx.size()/9; ++i)
    {
        const YsVec3 tri[3]=
        {
            YsVec3(vtx[i*9  ],vtx[i*9+1],vtx[i*9+2]),
            YsVec3(vtx[i*9+3],vtx[i*9+4],vtx[i*9+5]),
            YsVec3(vtx[i*9+6],vtx[i*9+7],vtx[i*9+8]),
        };
        YsPlane pln;
        pln.MakePlaneFromTriangle(tri[0],tri[1],tri[2]);

        YsVec3 itsc;
        if(YSOK==pln.GetIntersection(itsc,o,v))
        {
            auto side=YsCheckInsideTriangle3(itsc,tri);
            if(YSINSIDE==side || YSBOUNDARY==side)
            {
                auto dist=(itsc-o)*v; // Gives distance
                if(0.0<dist && (picked<0 || dist<pickedDist))
                {
                    picked=i;
                    pickedDist=dist;
                }
            }
        }
    }

    return picked;
}
```

8. Modify event handling for FSMOUSEEVENT_LBUTTONDOWN as:

```
if(evt==FSMOUSEEVENT_LBUTTONDOWN)
{
    MouseCoordinateTo3DLine(lastClick,mx,my);

    int triIdx=PickedTriangle(mx,my);
    if(0<=triIdx)
    {
        for(int i=0; i<3; ++i)
        {
            col[((3*triIdx)+i)*4  ]=1;
            col[((3*triIdx)+i)*4+1]=0;
            col[((3*triIdx)+i)*4+2]=0;
        }
    }
}
```

Picking a point

- What about picking a point? – Cannot calculate an intersection between a line and a point.
- 3D distance between a line and a point may not translate directly to the pixel distance.

Picking a point

- Transform a 3D coordinate into a screen coordinate and compare.

1. Add a member variable:

```
std::vector <float> pickedVtx;
```

2. Add PickedPoint function:

```
bool FsLazyWindowApplication::PickedPoint(YsVec3 &pos,int mx,int my) const
{
    int wid,hei;
    FsGetWindowSize(wid,hei);

    double pickedZ=YsInfinity;
    bool picked=false;
    YsVec3 pickedPos;
    for(int i=0; i<vtx.size()/3; ++i)
    {
        YsVec3 pos(vtx[i*3],vtx[i*3+1],vtx[i*3+2]);
        drawEnv.GetViewMatrix().Mul(pos,pos,1.0);
        drawEnv.GetProjectionMatrix().Mul(pos,pos,1.0);
        if(-1.0<=pos.z() && pos.z()<=1.0)
        {
            const double u=(pos.x()+1.0)/2.0;
            const double v=(pos.y()+1.0)/2.0;

            int x=(int)((double)wid*u);
            int y=hei-(int)((double)hei*v);
            if(mx-8<=x && x<=mx+8 && my-8<=y && y<=my+8)
            {
                if(true!=picked || pos.z()<pickedZ)
                {
                    pickedPos.Set(vtx[i*3],vtx[i*3+1],vtx[i*3+2]);;
                    pickedZ=pos.z();
                    picked=true;
                }
            }
        }
    }
    pos=pickedPos;
    return picked;
}
```

3. In Interval, add:

```
if(evt==FSMOUSEEVENT_RBUTTONDOWN)
{
    YsVec3 pos;
    if(true==PickedPoint(pos,mx,my))
    {
        pickedVtx.push_back(pos.x());
        pickedVtx.push_back(pos.y());
        pickedVtx.push_back(pos.z());
    }
}
```

4. In Draw function:

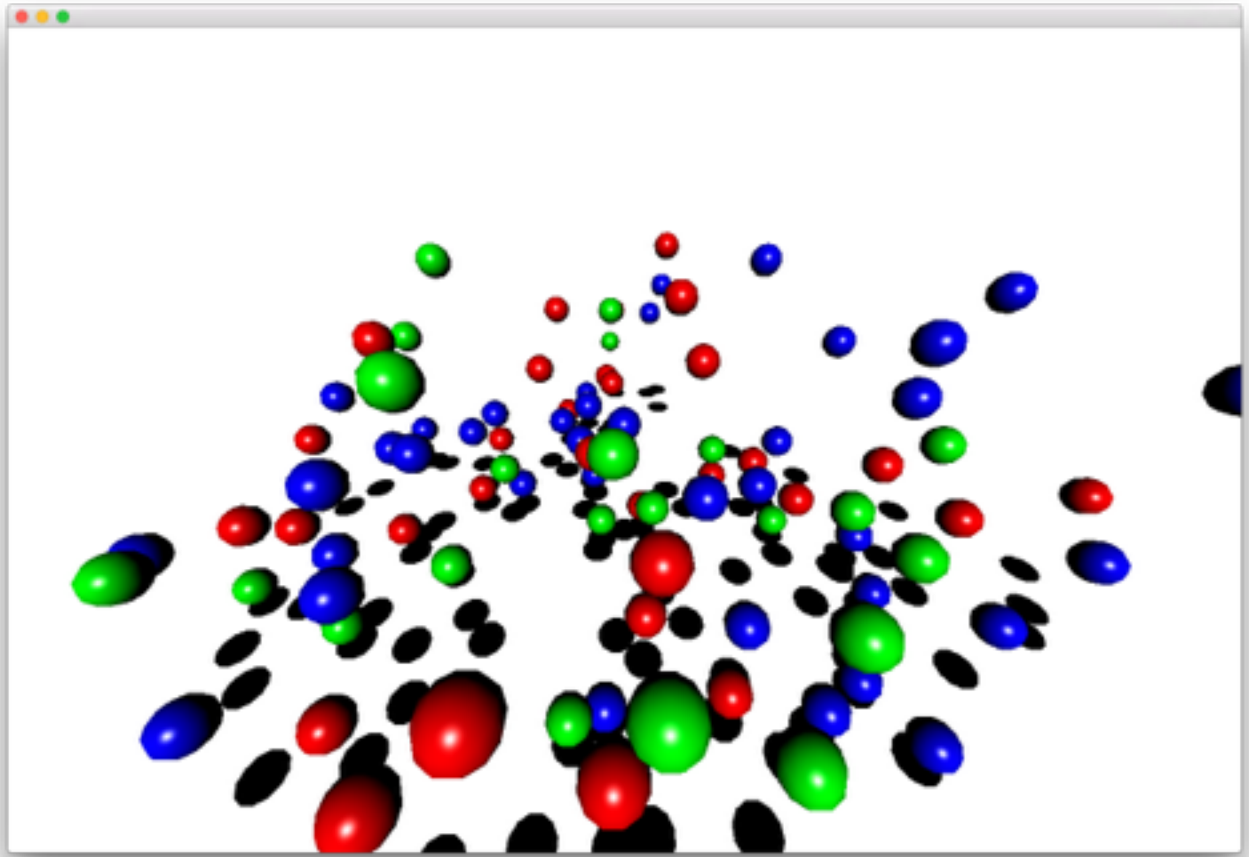
```
renderer.EnableZOffset();
renderer.SetZOffset(-0.00001);

GLfloat red[4]={1,0,0,1};
renderer.SetUniformColor(red);
glPointSize(4);
renderer.DrawVtx(GL_POINTS,pickedVtx.size()/3,pickedVtx.data());

renderer.DisableZOffset();
renderer.SetZOffset(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);
    }
}
```

Massive bouncing balls

- Let’s make 3D bouncing ball with 10000 balls.

1. Copy from bounce3d. Rename TARGET_NAME as glsl3d_bounce3d_with_lattice
2. Change NUM_BALL to 10000.
3. In the constructor,
   ```
   viewDist=80.0;
   ```
4. In SetInitialLocationAndVelocity, initial locations should be:
   ```
   int x=rand()%51-25;
   int y=25+rand()%51-25;
   int z=rand()%51-25;
   ```

5. In Move function, walls must be:

```
b.BounceOnWall(YsVec3( 60.0,0.0,0.0),YsVec3(-1.0,0.0,0.0));
b.BounceOnWall(YsVec3(-60.0,0.0,0.0),YsVec3( 1.0,0.0,0.0));
b.BounceOnWall(YsVec3(0.0, 60.0,0.0),YsVec3(0.0,-1.0,0.0));
b.BounceOnWall(YsVec3(0.0,-60.0,0.0),YsVec3(0.0, 1.0,0.0));
b.BounceOnWall(YsVec3(0.0,0.0, 60.0),YsVec3(0.0,0.0,-1.0));
b.BounceOnWall(YsVec3(0.0,0.0,-60.0),YsVec3(0.0,0.0, 1.0));
```

6. Shadow matrix must be:

```
YsMatrix4x4 shadowMat;
shadowMat.Translate(0,-60.0,0);
shadowMat.Scale(1.0,0.0,1.0);
```

Can see obvious drop of the frame rate.

3D Lattice

- Accelerates the collision detection.
- $O(N^2)$->$O(N)$ computational time.
- Use $O(N)$ memory.
- Very efficient for simulations with uniform (similar) particle size.

Other common data structure

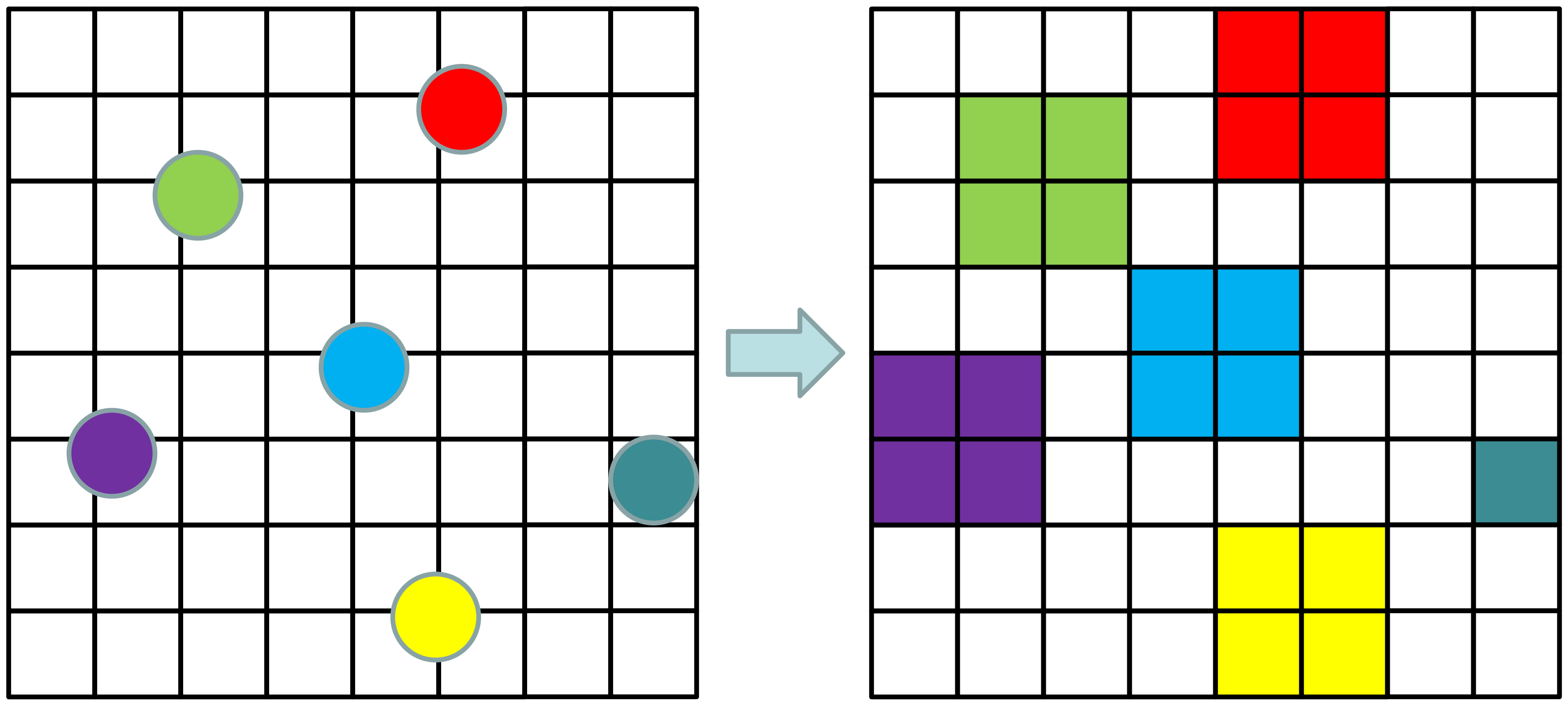
- Oct-tree (or octree)
- k-D tree

These data structures typically reduces the order of complexity to O(logN) even the point distribution is highly non-uniform. However, when points move, there is no easy update.

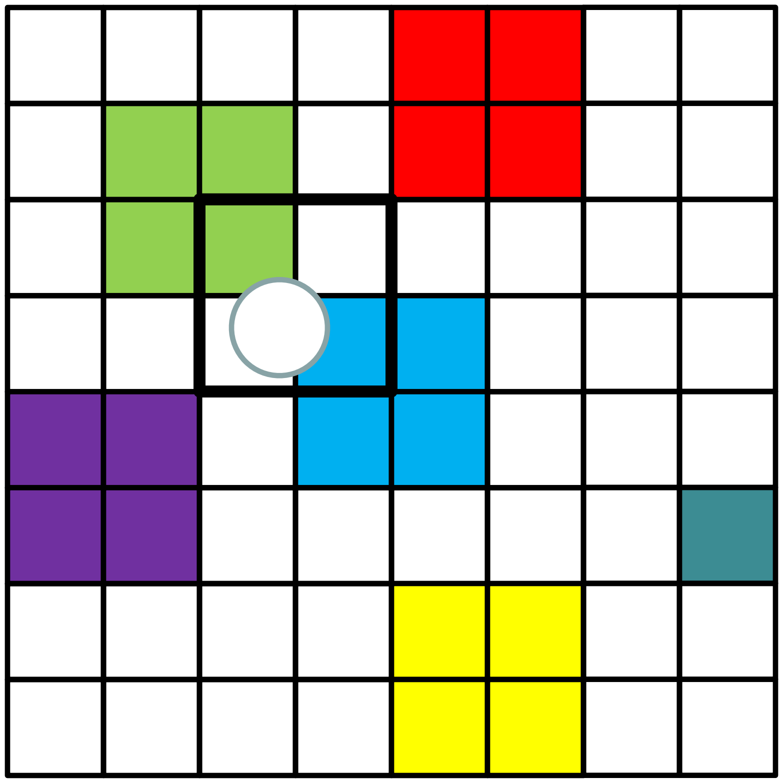
An advantage of lattice is its simplicity. Due to its simplicity, adding/removing of a primitive is very quick. When a point moves, just remove point, and re-register.

In many real situation, simplicity does a better job than a sophistication.

- You need to spend O(N) time to register balls in the lattice.



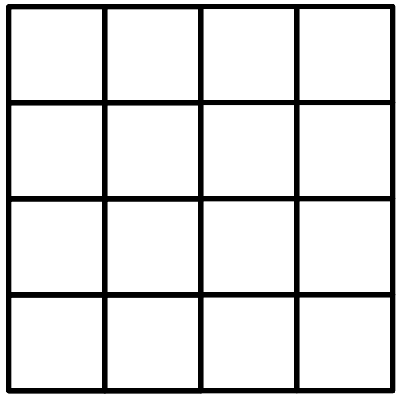
- For each ball, instead of checking collision against all other balls, the lattice will narrow down which other balls need to be checked for collision.



The white ball may be colliding with the green and blue balls, but not with other balls.

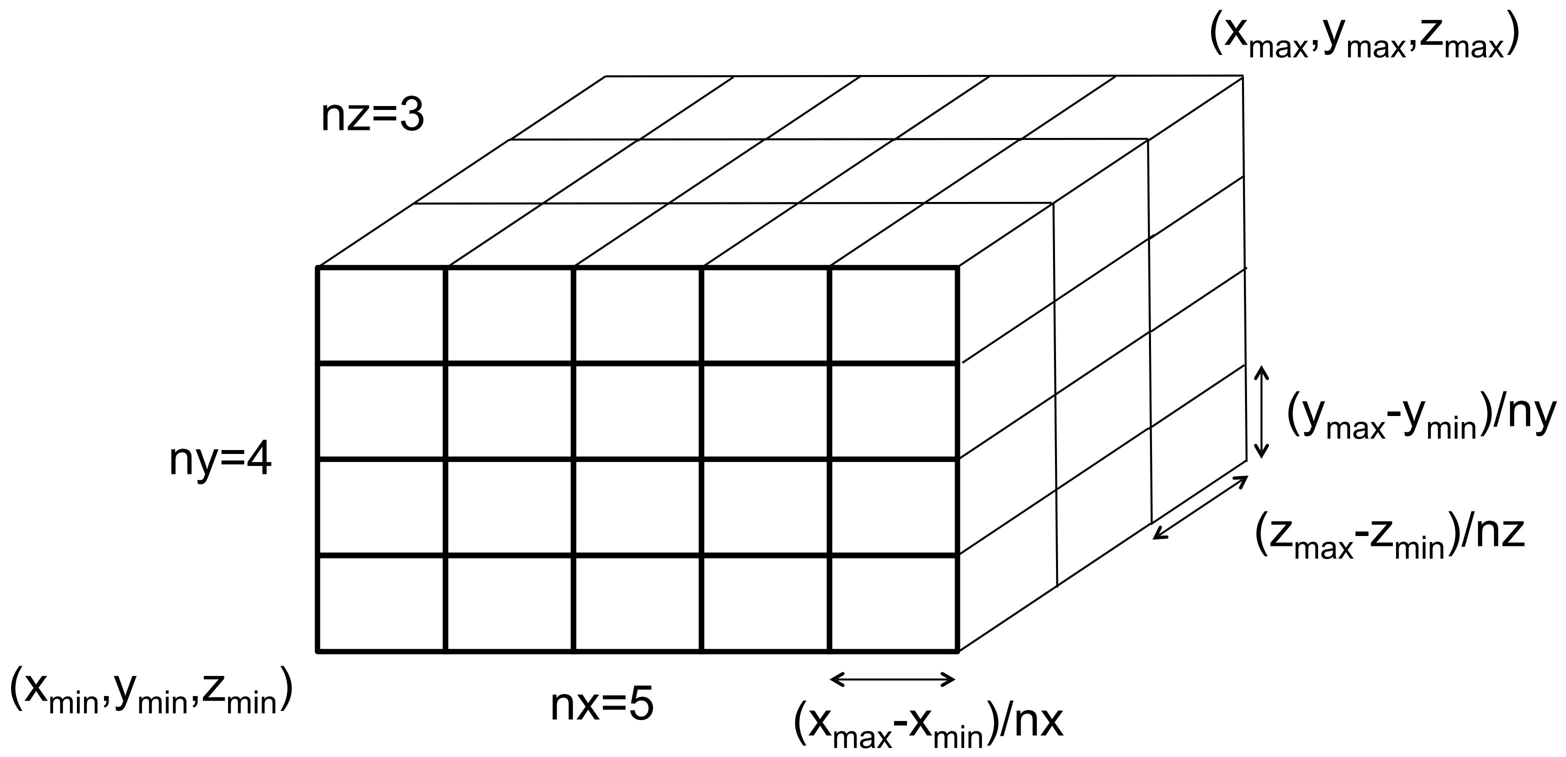
Lattice class

- Minimum information:
  - An array of the lattice elements.
  - Resolution nx,ny,nz.
- In this example, only cell information is necessary, but a general purpose lattice may need nodal information as well.
- A nx times ny times nz lattice needs (nx+1)*(ny+1)*(nz+1) nodes.



A 4x4 lattice has 5x5 nodes

Block Coord: (bx,by,bz) -> Linear Index: bz*(nx+1)*(ny+1)+by*(nx+1)+bx



Adding a lattice class – Minimum 3D lattice class

```
#ifndef LATTICE_IS_INCLUDED
#define LATTICE_IS_INCLUDED
/* { */

template <class T>
class Lattice3d
{
protected:
    int nx,ny,nz;
    T *dat;

private:
    Lattice3d(const Lattice3d <T> &incoming);
    Lattice3d &operator=(const Lattice3d <T> &incoming);
public:
    Lattice3d();
    ~Lattice3d();
    void CleanUp(void);

    T &Elem(int x,int y,int z);
    const T &Elem(int x,int y,int z) const;

    /*! nx,ny,nz=number of blocks, not nodes. */
    void Create(int nx,int ny,int nz);
};
template <class T>
Lattice3d<T>::Lattice3d()
{
    nx=0;
    ny=0;
    nz=0;
    dat=nullptr;
}
template <class T>
Lattice3d<T>::~Lattice3d()
{
    CleanUp();
}
```

```
template <class T>
void Lattice3d<T>::CleanUp(void)
{
    if(nullptr!=dat)
    {
        delete [] dat;
    }
    nx=0;
    ny=0;
    nz=0;
    dat=nullptr;
}
template <class T>
void Lattice3d<T>::Create(int nx,int ny,int nz)
{
    CleanUp();
    if(0<nx && 0<ny && 0<nz)
    {
        this->nx=nx;
        this->ny=ny;
        this->nz=nz;
        dat=new T [(nx+1)*(ny+1)*(nz+1)];
    }
}
template <class T>
T &Lattice3d<T>::Elem(int x,int y,int z)
{
    return dat[z*(nx+1)*(ny+1)+y*(nx+1)+x];
}
template <class T>
const T &Lattice3d<T>::Elem(int x,int y,int z) const
{
    return dat[z*(nx+1)*(ny+1)+y*(nx+1)+x];
}

/* } */
#endif
```

Make it a little more useful

```
template <class T>
class Lattice3d
{
protected:
    int nx,ny,nz;
    T *dat;
    YsVec3 min,max,dgn;

private:
    Lattice3d(const Lattice3d <T> &incoming);
    Lattice3d &operator=(const Lattice3d <T> &incoming);
public:
    Lattice3d();
    ~Lattice3d();
    void CleanUp(void);

    int GetN(void) const;
    T &Elem(int i);
    const T &Elem(int i) const;

    T &Elem(int x,int y,int z);
    const T &Elem(int x,int y,int z) const;
    /*! nx,ny,nz=number of blocks, not nodes. */
    void Create(int nx,int ny,int nz);

    void SetDimension(const YsVec3 &min,const YsVec3 &max);
    YsVec3i GetBlockIndex(const YsVec3 &pos);
    bool IsInRange(YsVec3i idx) const;
};

template <class T>
int Lattice3d<T>::GetN(void) const
{
    return (nx+1)*(ny+1)*(nz+1);
}
template <class T>
T &Lattice3d<T>::Elem(int i)
{
    return dat[i];
}
```

```
template <class T>
const T &Lattice3d<T>::Elem(int i) const
{
    return dat[i];
}

template <class T>
void Lattice3d<T>::SetDimension(const YsVec3 &min,const YsVec3
&max)
{
    this->min=min;
    this->max=max;
    this->dgn=max-min;
}
template <class T>
YsVec3i Lattice3d<T>::GetBlockIndex(const YsVec3 &pos) const
{
    const YsVec3 rel=pos-min;
    const double tx=rel.x()/dgn.x();
    const double ty=rel.y()/dgn.y();
    const double tz=rel.z()/dgn.z();
    int ix=(int)(tx*(double)nx);
    int iy=(int)(ty*(double)ny);
    int iz=(int)(tz*(double)nz);
    return YsVec3i(ix,iy,iz);
}
template <class T>
bool Lattice3d<T>::IsInRange(YsVec3i idx) const
{
    if(0<=idx.x() && idx.x()<=nx &&
       0<=idx.y() && idx.y()<=ny &&
       0<=idx.z() && idx.z()<=nz)
    {
        return true;
    }
    return false;
}
```

Use it from bouncing ball code.

1. Add:

```
#include "lattice.h"
```

2. Add member variable:

```
Lattice3d <std::vector <Ball *> > ltc;
```

3. In Initialize function:

```
ltc.Create(40,40,40);
ltc.SetDimension(YsVec3(-61.0,-61.0,-61.0),YsVec3(61.0,61.0,61.0));
```

4. Add:

```
double Ball::GetRadius(void) const
{
    return 1.0;
}
```

(Actually should have done earlier…)

5. Add RegisterBall function:

```
void FsLazyWindowApplication::RegisterBall(void)
{
    for(int i=0; i<ltc.GetN(); ++i)
    {
        ltc.Elem(i).clear();
    }

    for(auto &b : ball)
    {
        const auto r=b.GetRadius();
        const YsVec3 rv(r,r,r);
        const auto min=b.pos-rv,max=b.pos+rv;
        auto minIdx=ltc.GetBlockIndex(min);
        auto maxIdx=ltc.GetBlockIndex(max);
        for(auto x=minIdx.x(); x<=maxIdx.x(); ++x)
        {
            for(auto y=minIdx.y(); y<=maxIdx.y(); ++y)
            {
                for(auto z=minIdx.z(); z<=maxIdx.z(); ++z)
                {
                    if(true==ltc.IsInRange(YsVec3i(x,y,z)))
                    {
                        ltc.Elem(x,y,z).push_back(&b);
                    }
                }
            }
        }
    }
}
```

6. Modify BallCollision function:

```
void FsLazyWindowApplication::BallCollision(void)
{
    RegisterBall();
    for(int i=0; i<NUM_BALL; ++i)
    {
        const auto r=ball[i].GetRadius();
        const YsVec3 rv(r,r,r);
        const auto min=ball[i].pos-rv,max=ball[i].pos+rv;
        auto minIdx=ltc.GetBlockIndex(min);
        auto maxIdx=ltc.GetBlockIndex(max);
        for(auto x=minIdx.x(); x<=maxIdx.x(); ++x)
        {
            for(auto y=minIdx.y(); y<=maxIdx.y(); ++y)
            {
                for(auto z=minIdx.z(); z<=maxIdx.z(); ++z)
                {
                    if(true==ltc.IsInRange(YsVec3i(x,y,z)))
                    {
                        for(auto ballJPtr : ltc.Elem(x,y,z))
                        {
                            if(&ball[i]<ballJPtr)
                            {
                                auto &ball0=ball[i];
                                auto &ball1=*ballJPtr;

                                auto relVel=ball1.vel-ball0.vel;
                                auto relPos=ball1.pos-ball0.pos;
                                if(relPos.GetLength()<2.0 && relPos*relVel<0.0)
                                {
                                    auto wallPos=(ball0.pos+ball1.pos)/2.0;
                                    auto wallNom=YsUnitVector(ball1.pos-ball0.pos);
                                    ball1.BounceOnWall(wallPos,wallNom);
                                    ball0.BounceOnWall(wallPos,-wallNom);
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

How would you parallelize ball registration?

- In this case, calculating a cell index from ball coordinate is negligible.
- However, it is not always the case.
- Obviously the lattice is a shared resource in this situation.
- Must not be written to the same cell from multiple threads without locking.
- Locking will negate the benefit of parallel processing.
- Then how?