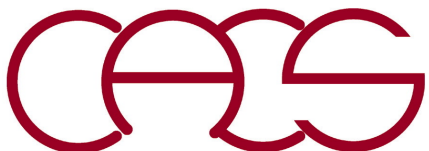# Visualizing Molecular Dynamics

Aiichiro Nakano

Collaboratory for Advanced Computing & Simulations
Department of Computer Science
Department of Physics & Astronomy
Department of Quantitative & Computational Biology
University of Southern California

Email: anakano@usc.edu

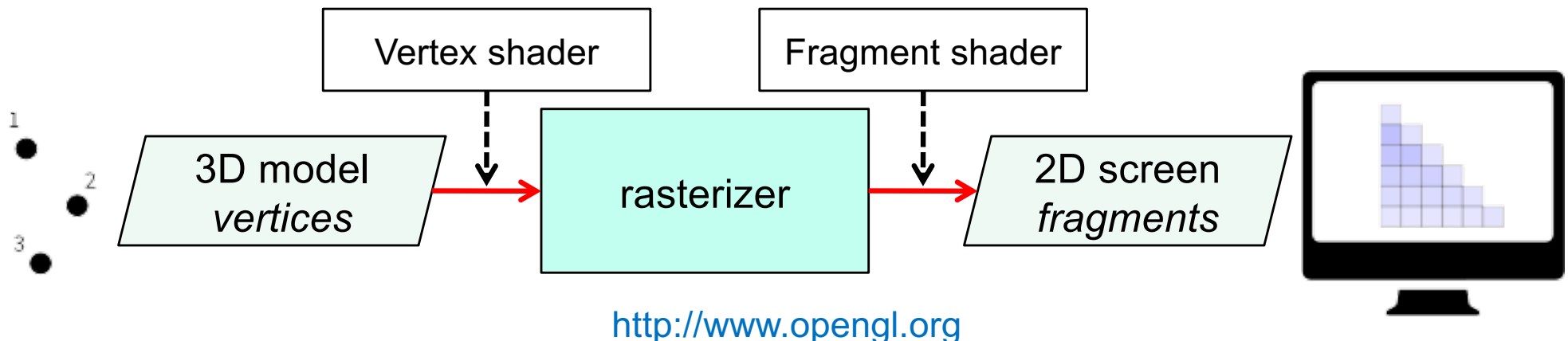**Goal: Visualize simulation to "understand" it**

OpenGL: Getting Started

Installing OpenGL & GLUT libraries:

- **OpenGL (Open Graphics Language): Standard, hardware-independent interface to graphics hardware.**

- **GLUT (OpenGL Utility Toolkit): Window-system-independent toolkit for window application programming interfaces (APIs).** **How to install OpenGL & GLUT:** http://web.eecs.umich.edu/~sugih/courses/eecs487/glut-howto

* **This lecture describes basic graphics concepts common with earlier versions of OpenGL. In newer versions, shaders written in the OpenGL Shading Language (GLSL) determine various rendering attributes.**

OpenGL rendering pipeline

http://www.opengl.org

OpenGL History

1992   **OpenGL released by Silicon Graphics, Inc. (SGI)**

1994   Beowulf cluster computer
MPI 1.0 (message passing)
`MPI_Send(), MPI_Recv()`
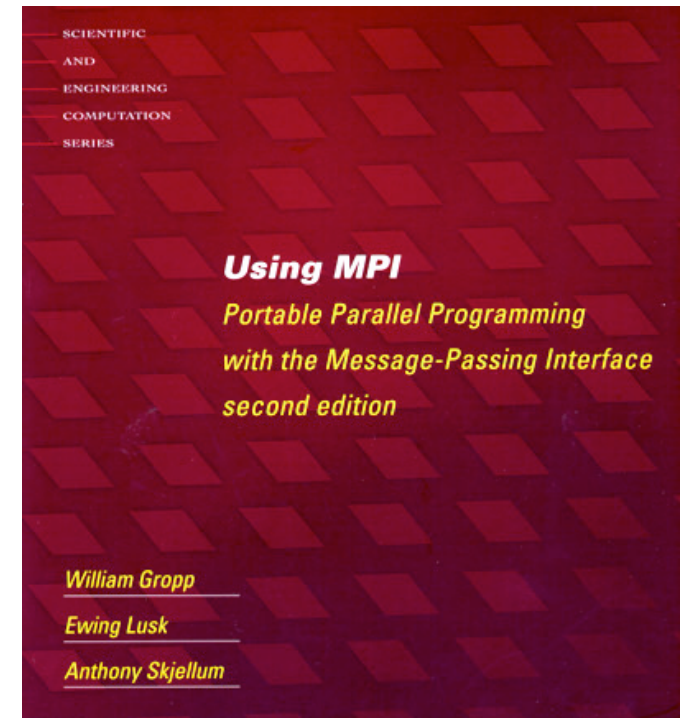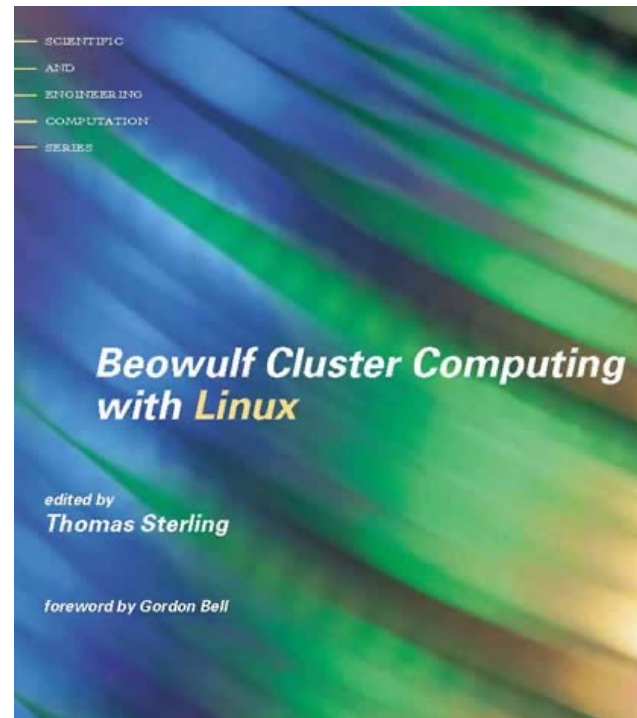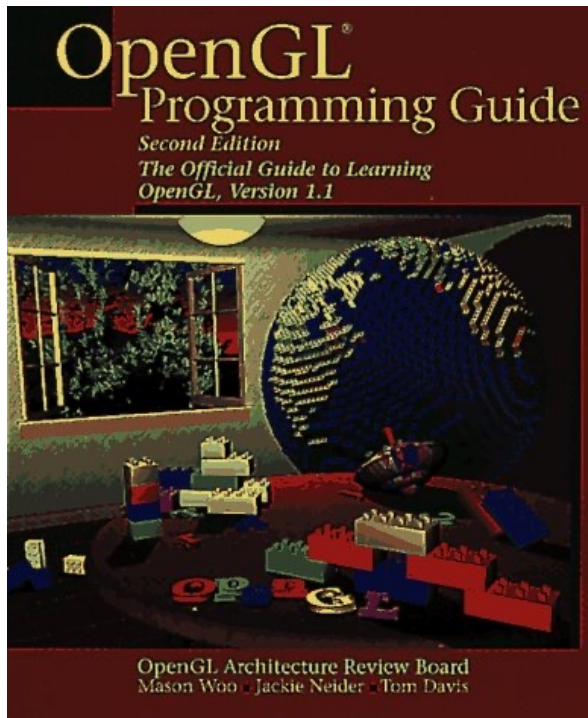
1997   OpenMP 1.0 (multithreading)
`#pragma omp parallel`

2004   **OpenGL2.0: OpenGL Shading Language (GLSL) included for controlling graphics cards**

2008   **OpenGL3.0: Major revision**

2007   CUDA 1.0 (graphics cards)

OpenGL® Programming Guide
Second Edition
The Official Guide to Learning OpenGL, Version 1.1
OpenGL Architecture Review Board
Mason Woo · Jackie Neider · Tom Davis

SCIENTIFIC AND ENGINEERING COMPUTATION SERIES
Beowulf Cluster Computing with Linux
edited by Thomas Sterling
foreword by Gordon Bell

SCIENTIFIC AND ENGINEERING COMPUTATION SERIES
Using MPI
Portable Parallel Programming with the Message-Passing Interface
second edition
William Gropp
Ewing Lusk
Anthony Skjellum

OpenGL Programming Basics
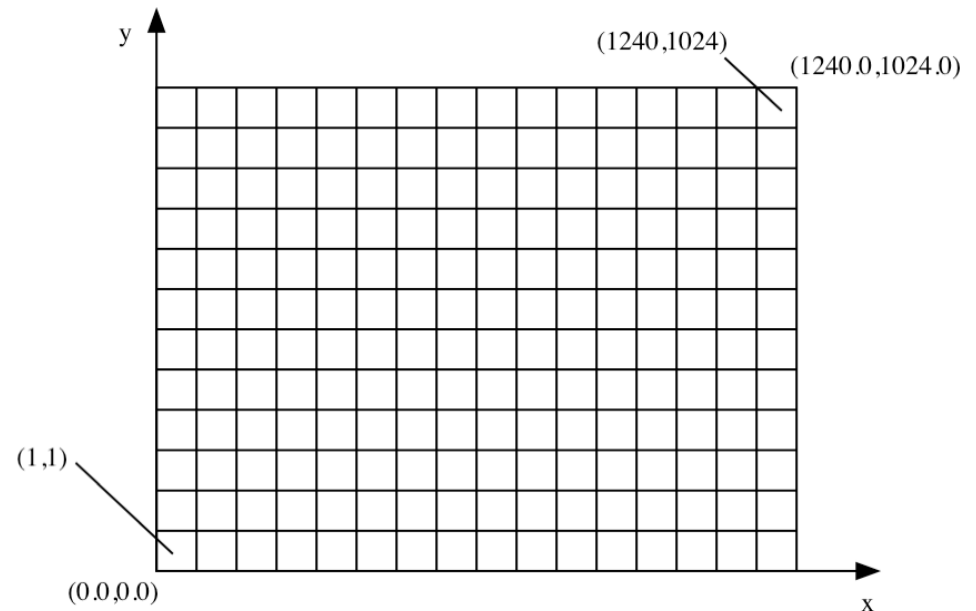
atomv.c ()

```c
#include <OpenGL/gl.h>      // Header File For The OpenGL Library
#include <OpenGL/glu.h>     // Header File For The GLu Library
#include <GLUT/glut.h>      // Header File For The GLut Library

glutInit(&argc, argv);

/* Set up an window */
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGBA|GLUT_DEPTH); /*Initialize display mode*/
glutInitWindowSize(winx, winy); /* Specify window size */
glutCreateWindow("Lennard-Jones Atoms"); /* Open window */
glEnable(GL_DEPTH_TEST); /* Enable z-buffer for hidden surface removal */
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT); /* Clear the window */
```

- **Frame buffer: A collection of buffers in memory, which store data for screen pixels (*e.g.*, 1280 pixels wide & 1024 pixels high) such as color, depth information for hidden surface removal, *etc*.**

OpenGL Event-Handling Loop

```
main() {
  /* Set a glut callback functions */
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);    events
  /* Start main display loop */
  glutMainLoop();
}


/* Definition of callback functions */
display() {...}
reshape() {...}    event handlers
```
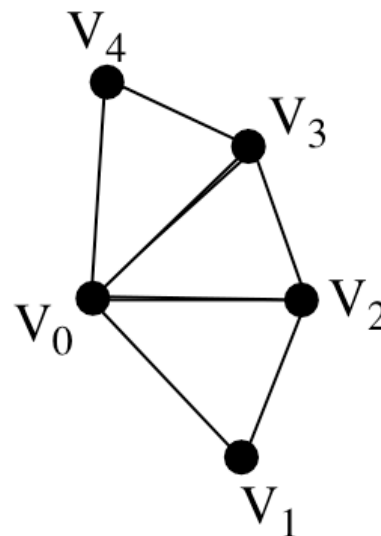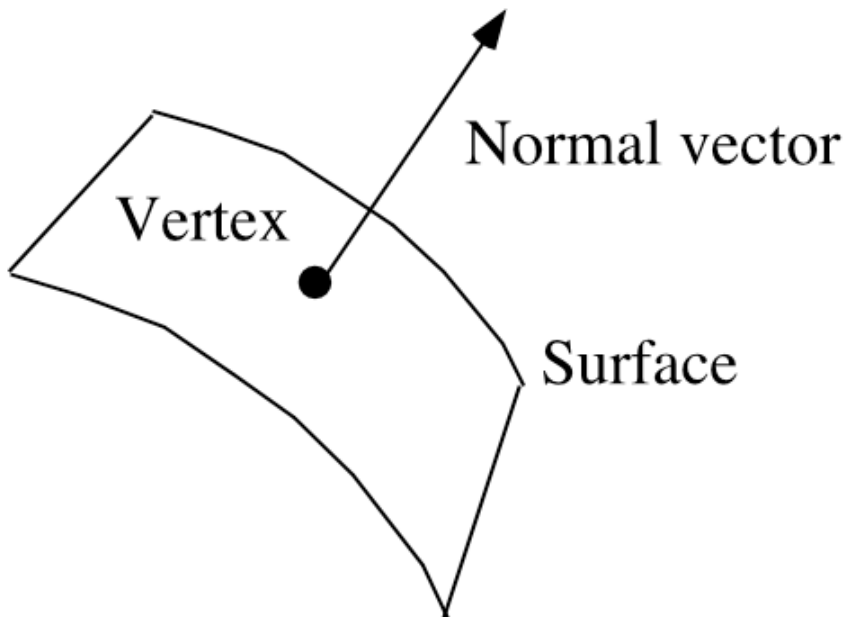
Glut runtime system keeps listening if any event happens; when an even happens, it invokes the corresponding user-specified event handler function.
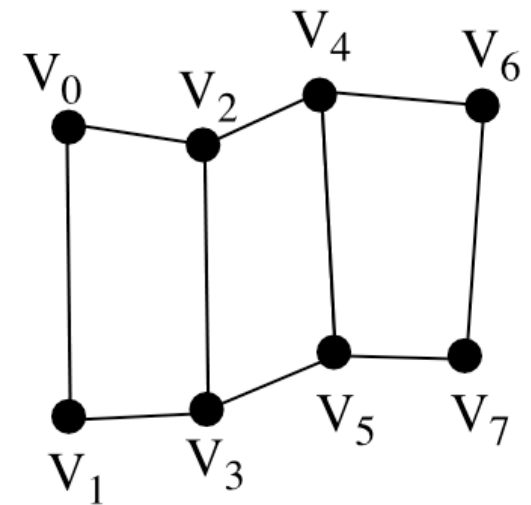
All topics in this presentation are detailed in
https://aiichironakano.github.io/cs596/Visual.pdf

Polygonal Surfaces

Drawing a set of polygons

```
float normal_vector[MAX_VERTICES][3],vertex_position[MAX_VERTICES][3];
glBegin(GL_QUAD_STRIP);
  for (i=0; i<number_of_vertices; i++) {
    glNormal3f(normal_vector[i]);
    glVertex3f(vertex_position[i]);
  }
glEnd();
```
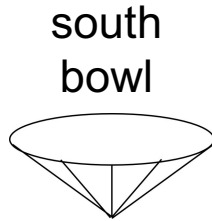

GL_TRIANGLE_FAN

GL_QUAD_STRIP

Polygonal Sphere

```c
int nlon=18, nlat=9;
loninc = 2*M_PI/nlon; /* Δφ */
latinc = M_PI/nlat;    /* Δθ */
/* South-pole triangular fan */
glBegin(GL_TRIANGLE_FAN);
  glNormal3f(0,-1,0);
  glVertex3f(0,-radius,0);
  lon = 0;
  lat = -M_PI/2 + latinc;
  y = sin(lat);
  for (i=0; i<=nlon; i++) {
    x = cos(lon)*cos(lat);
    z = -sin(lon)*cos(lat);
    glNormal3f(x,y,z);
    glVertex3f(x*radius,y*radius,z*radius);
    lon += loninc;}
glEnd();
/* Quadrilateral strips to cover the sphere */
for (j=1; j<nlat-1; j++) {
  lon = 0;
  glBegin(GL_QUAD_STRIP);
    for (i=0; i<=nlon; i++) {
      x = cos(lon)*cos(lat);
      y = sin(lat);
      z = -sin(lon)*cos(lat);
      glNormal3f(x,y,z);
      glVertex3f(x*radius,y*radius,z*radius);
      x = cos(lon)*cos(lat+latinc);
      y = sin(lat+latinc);
      z = -sin(lon)*cos(lat+latinc);
      glNormal3f(x,y,z);
      glVertex3f(x*radius,y*radius,z*radius);
      lon += loninc;}
  glEnd();
  lat += latinc;}
```
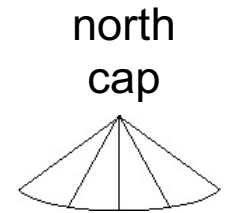
```c
/* North-pole triangular fan */
glBegin(GL_TRIANGLE_FAN);
  glNormal3f(0,1,0);
  glVertex3f(0,radius,0);
  y = sin(lat);
  lon = 0;
  for (i=0; i<=nlon; i++) {
    x = cos(lon)*cos(lat);
    z = -sin(lon)*cos(lat);
    glNormal3f(x,y,z);
    glVertex3f(x*radius,y*radius,z*radius);
    lon += loninc;
  }
glEnd();
```
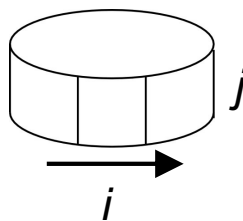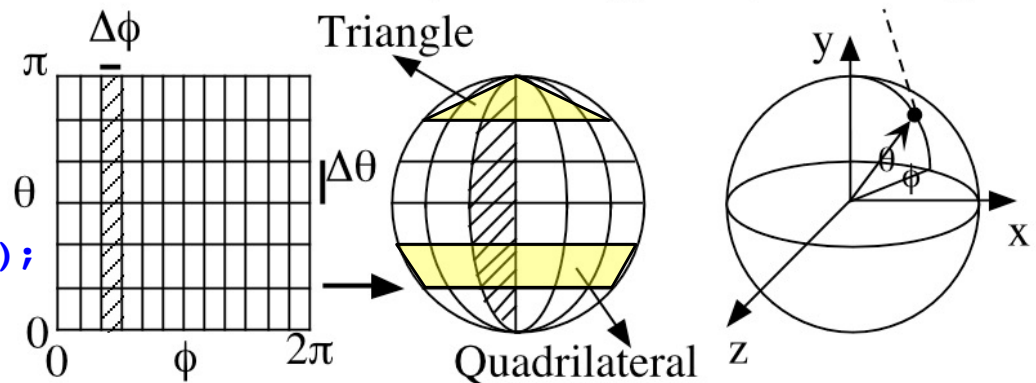
south bowl

north cap

Vertices in spherical → Cartesian coordinates

$(r\cos\theta\cos\phi, r\sin\theta, -r\cos\theta\sin\phi)$

Display Lists

- **Display list:** A group of OpenGL commands that have been stored for later execution.

```
/* Generates one new display-list ID */
GLuint sphereid = glGenLists(1);

/* Define a routine to draw a sphere*/
glNewList(sphereid, GL_COMPILE);
  ...code to draw a sphere (previous slide)...
glEndList();

/* Execute sphere drawing */
glCallList(sphereid);
```

Transformation Matrix

Drawing spheres at many atom positions

- **Transformation matrix:** Specifies the amount by which the object's coordinate system is to be rotated, scaled, or translated, *i.e.*, affine transformation.

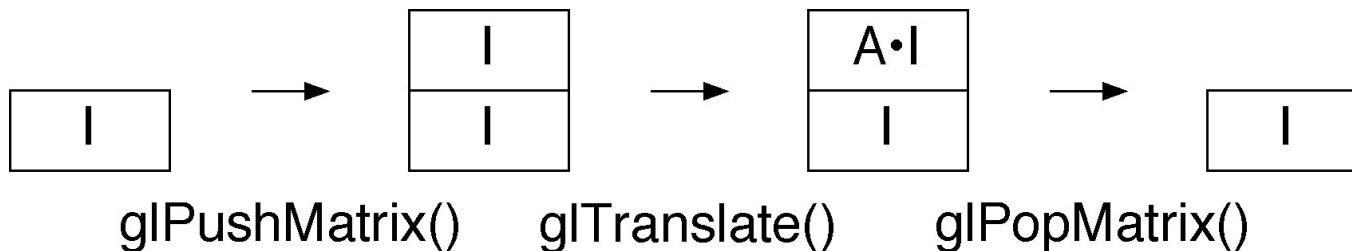$$\vec{r'} = \overleftrightarrow{A}\vec{r} + \vec{b}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

```
Matrix Identity
= {1, 0, 0, 0,
   0, 1, 0, 0,
   0, 0, 1, 0,
   0, 0, 0, 1};
```

- **Matrix stack:** A stack of transformation matrices—at the top of the stack is the current transformation matrix applied to all vertices. Initially the transformation matrix is the identity matrix.

```
glPushMatrix();
glTranslatef(atoms[i].crd[0],atoms[i].crd[1],atoms[i].crd[2]);
glCallList(sphereid);
glPopMatrix();
```
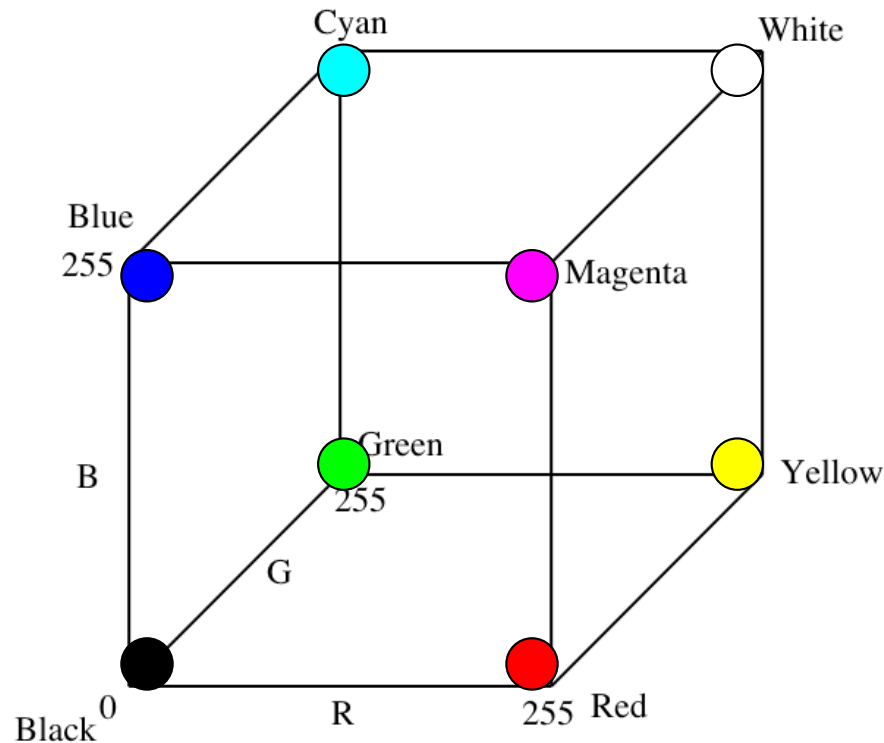
Repeat this *nAtom* times

glPushMatrix()   glTranslate()   glPopMatrix()

Color Display

- **RGB(A) mode: Specifying color by providing red, green & blue intensities (& alpha component).**
- **Alpha component: Specifies the opacity of a material; default value is 1.0 (nontransparent), if not specified.**

```
float r=1.0, g=0.0, b=0.0;

glColor3f(r,g,b);
```
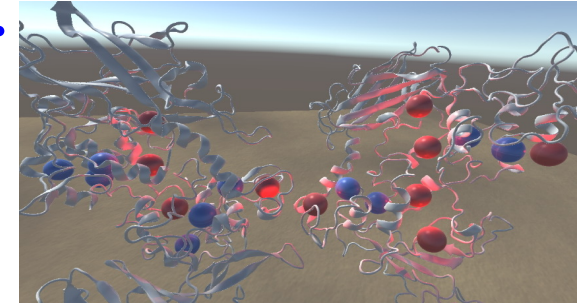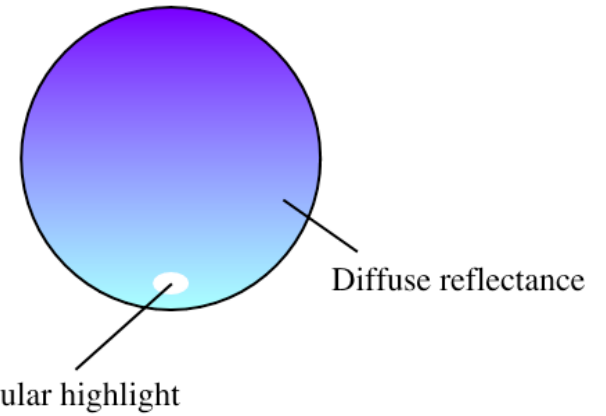

- **OpenGL as a state machine: Color change stays.**

Lighting & Materials

**OpenGL color =
light × material-reflectance**

OpenGL Color Types

- **Diffuse component:** Gives the appearance of a matter or flat reflection from an object's surface.
- **Ambient illumination:** Simulates light reflected from other objects.
- **Specular light:** Creates highlights.
- **Emission:** Simulates the appearance of lights in the scene.

Diffuse reflectance

Specular highlight

Materials Definition

- **Refelectance:** Material is characterized by ambient, diffuse & specular reflectance, *i.e.*, how the object reflects light.
- **glEnable(GL_COLOR_MATERIAL)**
  In this mode, the current color specified by **glColor*()** will change the ambient & diffuse reflectance.
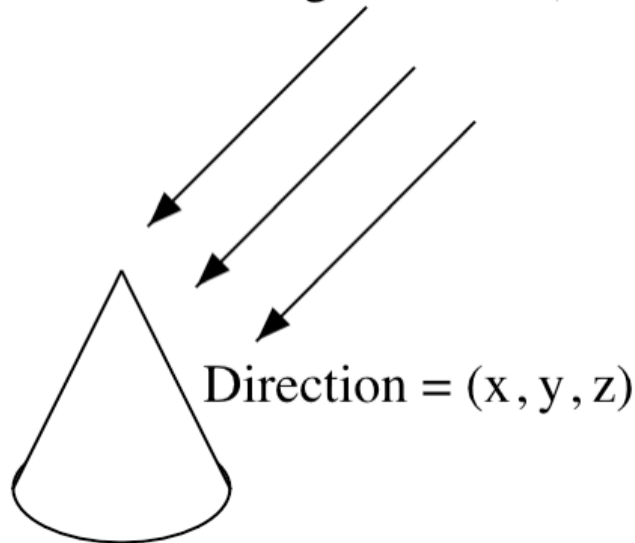
Lighting Source

color = light × material (*e.g.,* $\alpha = \alpha_{\text{light}} \times \alpha_{\text{material}}$)

```
float light_diffuse[4] = {1.0,1.0,1.0,1.0};
float light_position[4] = {0.5,0.5,1.0,0.0};

/* Define a lighting source */
glLightfv(GL_LIGHT0,GL_DIFFUSE,light_diffuse);
glLightfv(GL_LIGHT0,GL_POSITION,light_position);

/* Enable a single OpenGL light */
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
```
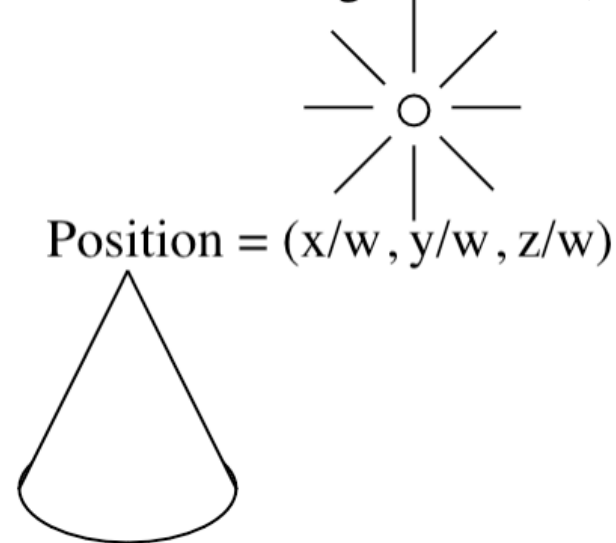
Directional light source (w = 0)   Point light source (w ≠ 0)

Position = (x/w, y/w, z/w)

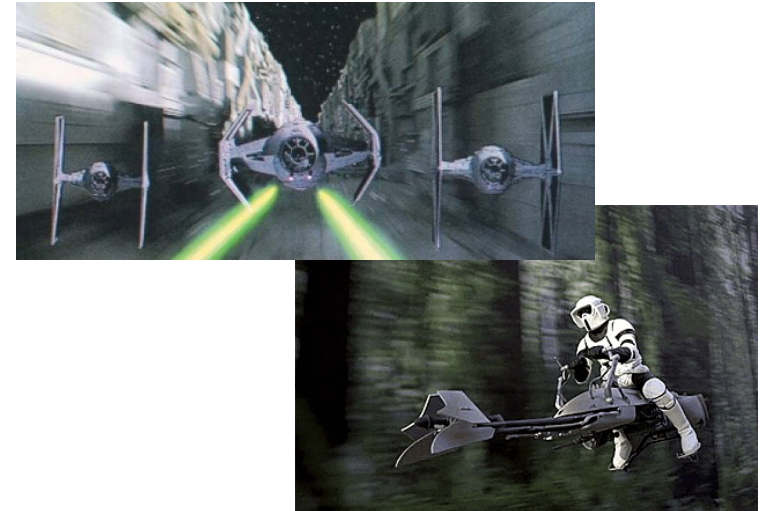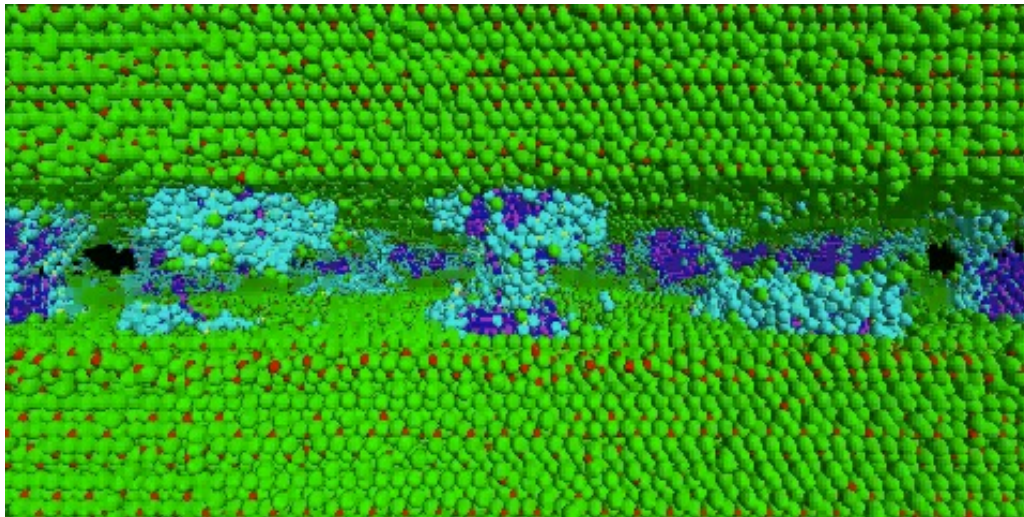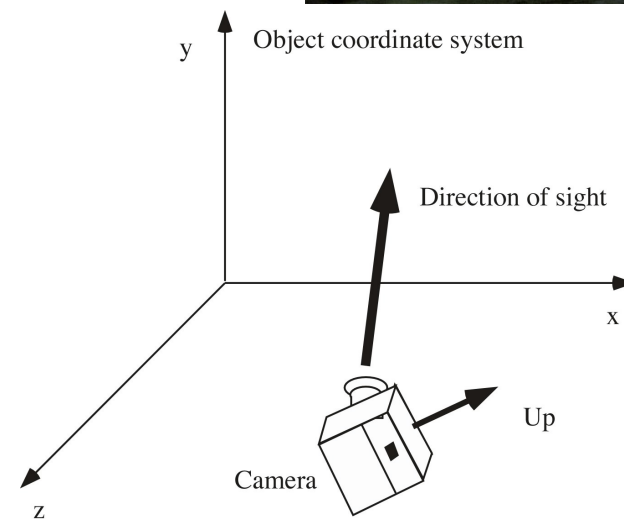Direction = (x, y, z)

Viewing Transformation

- **Viewing transformation:** Transforms object coordinates to eye coordinates.

`gluLookat(eyx, eyey, eyz, centerx, centery, centerz, upx, upy, upz);`

Eye coordinate system

Camera specification

Clipping

```
void reshape (int w, int h) {
  ...
  /* set the GL viewport to match the full size of the window */
  glViewport(0, 0, (GLsizei)w, (GLsizei)h);
  aspect = w/(float)h;
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(fovy,aspect,near_clip,far_clip);
  glMatrixMode(GL_MODELVIEW);
}
```

$$fovy = 2\tan^{-1}\left(\frac{L_y/2}{dis - L_z/2}\right)$$

View Frustum

$$dis = \sqrt{L_x^2 + L_y^2 + L_z^2}$$

Let frustum contain entire simulation box vertically

Matrix Modes

glMatrixMode(*GL_MODELVIEW* | *GL_PROJECTION*)

gluLookat()   *gluPerspective*()

Animation

```
main() {
    glutIdleFunc(animate);
}
...
void animate() { /* Callback function for idle events */
/* Keep updating the scene until the last MD step is reached */
    if (stepCount <= StepLimit) {
        SingleStep(); /* One MD-step integration */
        if (stepCount%StepAvg == 0) EvalProps();
        makeCurframeGeom(); /* Redraw the scene (make a display list) */
        glutPostRedisplay(); Force display event ~ user request for display() call
        ++stepCount;
    }
}
```
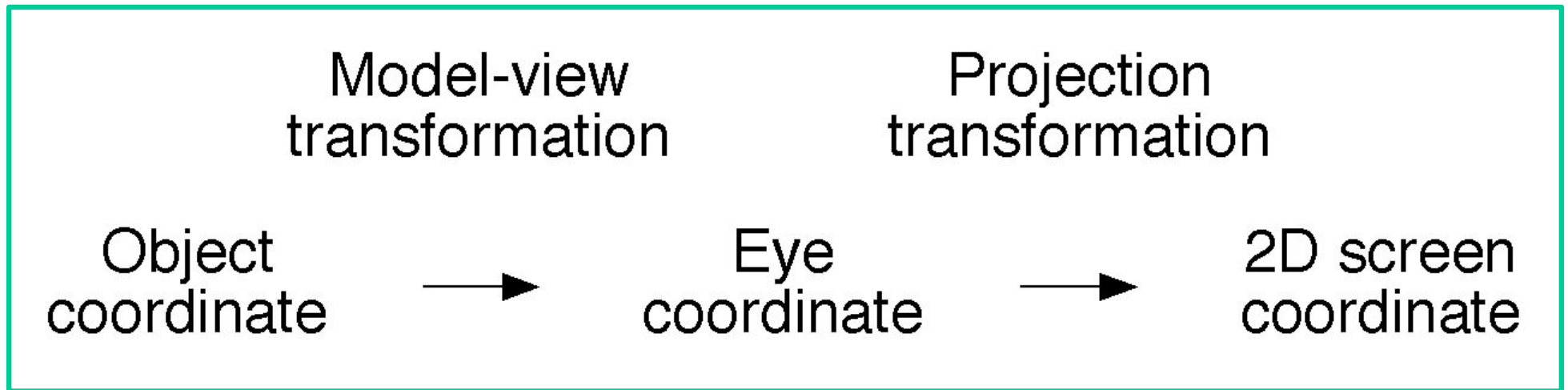
"idle" event handler: Called when current scene has been drawn & nothing to do

Advance simulation one time-step

Rendering occurs in

Back buffer

Screen

User sees

Front buffer

Read

Call display list to render all atoms

```
void display() {
    ...
    drawScene();
    glutSwapBuffers();
}
```

Exchange front & back buffers

Do-It-Yourself MD Animation

- **Combine** `md.c` (https://aiichironakano.github.io/cs596/src/md/) **&** `atomv.c` (https://aiichironakano.github.io/cs596/src/viz/) **to write a C+OpenGL program for** *in situ* **animation of simulation, following the lecture note on "***Visualizing Molecular Dynamics III—Animation***":**
  https://aiichironakano.github.io/cs596/Visual.pdf

1. Initialize atomic coordinates, velocities, accelerations & step count before entering the GLUT main loop in main()
   InitParams();   // Read and initialize MD parameters
   InitConf();
   ComputeAccel(); // Compute initial accelerations
   stepCount = 1; // Initialize the MD step count
2. Register single-MD-step update function, say animate(), as glutIdleFunc() callback function
   glutIdleFunc(animate);
3. In the callback function, animate()
   if stepCount ≤ StepLimit
      (i)   velocity-Verlet integration for one MD step, SingleStep();
      (ii) make a display list for a collection of atoms with the updated atomic coordinates, *i.e.*, call makeCurframeGeom();
      (iii) call glutPostRedisplay() to redraw the updated scene;
      (iv) increment the time step, stepCount;
   endif

Do-It-Yourself MD Animation (2)

4. Base program is atomv.c (it will be an OpenGL program)

Rename atomv.h|c to mdv.h|c

Copy (a) all contents of md.h into mdv.h & (b) all functions but main() from md.c to mdv.c; note most contents of main() in md.c have been used in animate() & main() in mdv.c in the previous slides

atomv.h	md.h
~~int natoms~~	int nAtom
~~AtomType *atoms~~	double r[NMAX][3]

Only used in `makeAtoms()` — replace by `md.h` counterparts

```
for (i=0; i < natoms; i++) {                        makeAtoms() in atomv.c
  glPushMatrix();
  glTranslatef(atoms[i].crd[0],atoms[i].crd[1],atoms[i].crd[2]);
  glColor3f(rval,gval,bval);
  glCallList(sphereid);
  glPopMatrix();
}
```

```
for (i=0; i < nAtom; i++) {                         makeAtoms() in mdv.c
  glPushMatrix();
  glTranslatef(r[i][0],r[i][1],r[i][2]);
  glColor3f(rval,gval,bval);
  glCallList(sphereid);
  glPopMatrix();
}
```

Animation for final Project?

- "Splash in a flash: sharpness-aware minimization for efficient liquid splash simulation," V. Jetly, H. Ibayashi, & A. Nakano, *Eurographics*, 1003 ('22)

- "GPU-based inverse rendering with multi-objective particle swarm optimization," K. Nagano, T. Collins, C.-A. Chen & A. Nakano, *SIGGRAPH Asia* ('15)

Ours: MLFLIP+SAM

MLFLIP
(Same training epochs)