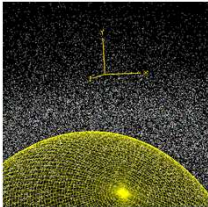**OpenCL / OpenGL Vertex Buffer Interoperability:**
A Particle System Case Study

Also, see the video at:
http://cs.oregonstate.edu/~mjb/cs575/Projects/particles.mp4

Oregon State University

Mike Bailey

mjb@cs.oregonstate.edu

Oregon State University
Computer Graphics

opencl.opengl.vbo.pptx
mjb – March 27, 2021

OpenCL / OpenGL Vertex Interoperability: The Basic Idea

Your C++ program writes initial values into the buffer on the GPU

OpenCL acquires the buffer

Each OpenCL kernel reads an (x,y,z) value from the buffer

Each OpenCL kernel updates its (x,y,z) value

(x,y,z) Vertex Data in an OpenGL Buffer

Each OpenCL kernel writes its (x,y,z) value back to the buffer

OpenCL releases the buffer

OpenGL draws using the (x,y,z) values in the buffer on the GPU

Oregon State University
Computer Graphics
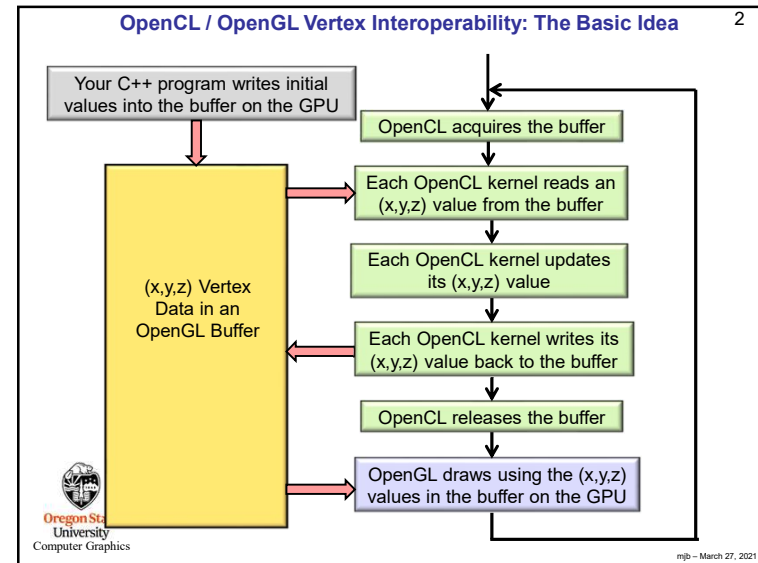
mjb – March 27, 2021

Some of the Inner Workings of OpenGL:
Feel Free to Detour Right to Slide #24 if You Don't Want to Know This

DETOUR

Oregon State University
Computer Graphics

mjb – March 27, 2021
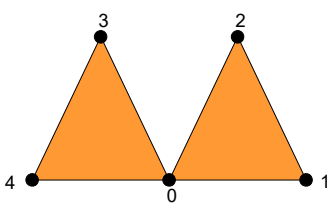
In the Beginning of OpenGL …

You listed the vertices with separate function calls:

```
glBegin( GL_TRIANGLES );
        glVertex3f( x0, y0, z0 );
        glVertex3f( x1, y1, z1 );
        glVertex3f( x2, y2, z2 );

        glVertex3f( x0, y0, z0 );
        glVertex3f( x3, y3, z3 );
        glVertex3f( x4, y4, z4 );
glEnd( );
```

Then someone noticed how inefficient that was, for three reasons:

1. **Sending large amounts of small pieces of information is less efficient than sending small amounts of large pieces of information**

2. **The vertex coordinates were being listed in the CPU and were being transferred to the GPU every drawing pass**

3. **Some vertices were listed twice**

Oregon State University
Computer Graphics

mjb – March 27, 2021

1

Here's What OpenGL Has Been Moving To: Vertex Buffer Objects

```
GLfloat CubeVertices[ ][3] =
{
        { -1., -1., -1. },
        {  1., -1., -1. },
        { -1.,  1., -1. },
        {  1.,  1., -1. },
        { -1., -1.,  1. },
        {  1., -1.,  1. },
        { -1.,  1.,  1. },
        {  1.,  1.,  1. }
};
```

```
GLfloat CubeColors[ ][3] =
{
        { 0., 0., 0. },
        { 1., 0., 0. },
        { 0., 1., 0. },
        { 1., 1., 0. },
        { 0., 0., 1. },
        { 1., 0., 1. },
        { 0., 1., 1. },
        { 1., 1., 1. },
};
```

```
GLuint CubeIndices[ ][4] =
{
        { 0, 2, 3, 1 },
        { 4, 5, 7, 6 },
        { 1, 3, 7, 5 },
        { 0, 4, 6, 2 },
        { 2, 6, 7, 3 },
        { 0, 1, 5, 4 }
};
```

Oregon State
University
Computer Graphics

mjb – March 27, 2021

A Little Background -- the OpenGL *Rendering Context*

The OpenGL Rendering Context contains all the characteristic information necessary to produce an image from geometry. This includes transformations, colors, lighting, textures, where to send the display, etc.

Array Buffer   Element Array Buffer   Texture0   Texture1   Display Dest.
Context
Color   Lighting   Transformation

If we were implementing the OpenGL state as a C++ structure (which we're not), we might do something like this:

```
struct context
{
        float [4]                    Color;
        float [4][4]                 Transformation;
        struct Texture *             Texture0;
        struct DataArrayBuffer *     ArrayBuffer;
        . . .
} Context;
```

Oregon State
University
Computer Graphics

mjb – March 27, 2021

More Background –
How do you create a special OpenGL Array Buffer called a Vertex Buffer Object?

In C++, objects are pointed to by their address.

In OpenGL, objects are pointed to by an unsigned integer handle. You can assign a value for this handle yourself (not recommended), or have OpenGL generate one for you that is guaranteed to be unique. For example:

```
GLuint buf;

glGenBuffers( 1, &buf );
```

This doesn't actually allocate memory for the buffer object yet, it just acquires a unique handle. To allocate memory, you need to bind this handle to the Context.

Oregon State
University
Computer Graphics

mjb – March 27, 2021

More Background – What is an OpenGL "Object"?

An OpenGL Object is pretty much the same as a C++ object: it encapsulates a group of data items and allows you to treat them as a unified whole. For example, a Data Array Buffer Object *could* be defined in C++ by:

```
struct  DataArrayBuffer
{
        enum      dataType;
        void *    memStart;
        int       memSize;
};
```

Then, you could create any number of Buffer Object instances, each with its own characteristics encapsulated within it. When you want to make that combination current, you just need to point the ArrayBuffer element of the Context to that entire struct **("bind")**. When you bind an object, all of its information comes with it.

Oregon State
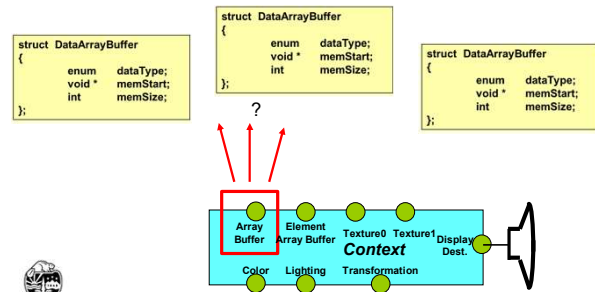University
Computer Graphics

mjb – March 27, 2021

A Little Background -- the OpenGL *Rendering Context*

It's very fast to re-bind a different vertex buffer. It amounts to just changing a pointer.
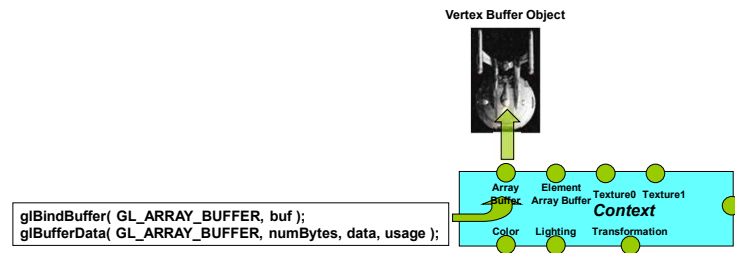
```
glBindBuffer( GL_ARRAY_BUFFER, buf );
```

```
struct DataArrayBuffer
{
    enum     dataType;
    void *   memStart;
    int      memSize;
};
```

```
struct DataArrayBuffer
{
    enum     dataType;
    void *   memStart;
    int      memSize;
};
```

```
struct DataArrayBuffer
{
    enum     dataType;
    void *   memStart;
    int      memSize;
};
```

?

Array Buffer | Element Array Buffer | Texture0 | Texture1 | Display Dest.
Context
Color | Lighting | Transformation

Oregon State University
Computer Graphics

mjb – March 27, 2021

More Background -- "Binding" to the Context

The OpenGL term "binding" refers to "attaching" or "docking" (a metaphor which I find to be more visually pleasing) an OpenGL object to the Context. You can then assign characteristics, and they will "flow" through the Context into the object.

Vertex Buffer Object

```
glBindBuffer( GL_ARRAY_BUFFER, buf );
glBufferData( GL_ARRAY_BUFFER, numBytes, data, usage );
```

Array Buffer | Element Array Buffer | Texture0 | Texture1
Context
Color | Lighting | Transformation

Think of it as happening this way:

```
Context.ArrayBuffer.memStart = CopyToGpuMemory( data, numBytes );
Context.ArrayBuffer.memSize = numBytes;
```
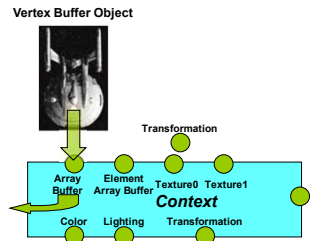
Oregon State University
Computer Graphics

mjb – March 27, 2021

More Background -- "Binding" to the Context

When you want to *use* that Vertex Buffer Object, just bind it again. All of the characteristics will then be active, just as if you had specified them again.

Vertex Buffer Object

Transformation

Array Buffer | Element Array Buffer | Texture0 | Texture1
Context
Color | Lighting | Transformation

```
glBindBuffer( GL_ARRAY_BUFFER, buf );
```

Think of it as happening this way:

```
float *data = Context.ArrayBuffer.memStart;
```

Oregon State University
Computer Graphics

mjb – March 27, 2021

Vertex Buffers: Putting Data in the Buffer Object

```
glBufferData( type,  numBytes,  data,  usage );
```

type is the type of buffer object this is:
GL_ARRAY_BUFFER to store floating point vertices, normals, colors, and texture coordinates

numBytes is the number of bytes to store in all. Not the number of numbers, but the number of *bytes*!

data is the memory address of (i.e., pointer to) the data to be transferred to the graphics card. *This can be NULL, and the data can be transferred later via memory-mapping.*

Oregon State University
Computer Graphics

mjb – March 27, 2021

Preview: We are going to use a Particle System as a Case Study

Your C++ program writes initial values into the buffer on the GPU

OpenCL acquires the buffer

Each OpenCL kernel reads an (x,y,z) value from the buffer

Each OpenCL kernel updates its (x,y,z) value

Each OpenCL kernel writes its (x,y,z) value back to the buffer

OpenCL releases the buffer

OpenGL draws using the (x,y,z) values in the buffer on the GPU

(x,y,z) Vertex Data in an OpenGL Buffer

Oregon State University
Computer Graphics

mjb – March 27, 2021

13

Vertex Buffers: Putting Data in the Buffer Object

glBufferData(type, numbytes, data, usage);

usage is a hint as to how the data will be used: GL_xxx_yyy

where xxx can be:

STREAM	this buffer will be written lots
STATIC	this buffer will be written seldom and read often
DYNAMIC	this buffer will be written often and used often

and yyy can be:

DRAW	this buffer will be used for drawing
READ	this buffer will be copied into
COPY	not a real need for now, but someday…

GL_STATIC_DRAW is the most common usage

Oregon State University
Computer Graphics

mjb – March 27, 2021

14

Vertex Buffers: Step #1 – Fill the Arrays

```
GLfloat  Vertices[ ][3] =
{
          { 1.,  2.,  3. },
          { 4.,  5.,  6. },
          . . .
};


int numVertices = sizeof(Vertices)  /  ( 3*sizeof(GLfloat) );
```

Oregon State University
Computer Graphics

mjb – March 27, 2021

15

Vertex Buffers: Step #2 – Create the Buffers and Fill Them

```
glGenBuffers( 1, &buf );

glBindBuffer( GL_ARRAY_BUFFER, buf );
glBufferData( GL_ARRAY_BUFFER, 3*sizeof(GLfloat)*numVertices, Vertices, GL_STATIC_DRAW );
```

Oregon State University
Computer Graphics

mjb – March 27, 2021

16

Slide 17

glEnableClientState(type)

where *type* can be any of:

```
GL_VERTEX_ARRAY
GL_COLOR_ARRAY
GL_NORMAL_ARRAY
GL_TEXTURE_COORD_ARRAY
```

• Call this as many times as you need to enable all the arrays that you will need.

• There are other types, too.

• To deactivate a type, call:
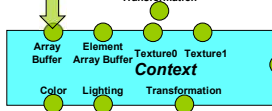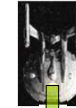
glDisableClientState(type)

Oregon State
University
Computer Graphics

mjb – March 27, 2021

17

Slide 18

glBindBuffer(GL_ARRAY_BUFFER, buf);

Vertex Buffer Object

Transformation

Array Buffer | Element Array Buffer | Texture0 | Texture1
Context

Color | Lighting | Transformation

Oregon State
University
Computer Graphics

mjb – March 27, 2021

18

Slide 19

```
glVertexPointer( size, type, stride, rel_address );
glColorPointer( size, type, stride, rel_address );
glNormalPointer( type, stride, rel_address );
glTexCoordPointer( size, type, stride, rel_address );
```

Vertex Data

Color Data

vs.

Vertex Data
Color Data
Vertex Data
Color Data
Vertex Data
Color Data

size is the spatial dimension, and can be: 2, 3, or **4**

type can be:
```
GL_SHORT
GL_INT
GL_FLOAT
GL_DOUBLE
```

stride is the byte offset between consecutive entries in the array (**0** means tightly packed)

rel_address, the 4th argument, is the relative byte address from the start of the buffer where the first element of this part of the data lives.

Oregon State
University
Computer Graphics

mjb – March 27, 2021

19

Slide 20

gl*Pointer(size, type, stride, offset);

rel_address, the 4th argument, is the relative byte address from the start of the buffer where the first element of this part of the data lives.

Packed:

glVertexPointer(3, GL_FLOAT, 3*sizeof(GLfloat), 0);

glColorPointer(3, GL_FLOAT, 3*sizeof(GLfloat), 3*numVertices*sizeof(GLfloat));

Vertex Data

Color Data

stride   start

Interleaved:

glVertexPointer(3, GL_FLOAT, 6*sizeof(GLfloat), 0);

glColorPointer(3, GL_FLOAT, 6*sizeof(GLfloat), 3*sizeof(GLfloat));

Vertex Data
Color Data
Vertex Data
Color Data
Vertex Data
Color Data

Oregon State
University
Computer Graphics

mjb – March 27, 2021

20

Vertex Buffers: Step #6 – Specify the Connections

```
GLfloat Vertices[ ][3] =
{
        { x₀,  y₀,  z₀ },
        { x₁,  y₁,  z₁ },
        { x₂,  y₂,  z₂ },
        { x₃,  y₃,  z₃ },
        { x₄,  y₄,  z₄ },
        { x₅,  y₅,  z₅ }
};


int  numVertices = sizeof(Vertices)  /  ( 3*sizeof(GLfloat) );


glDrawArrays( GL_TRIANGLES, 0, numVertices );
```


Oregon State
University
Computer Graphics

mjb – March 27, 2021

Vertex Buffers: Writing Data Directly into a Vertex Buffer

Map the buffer from GPU memory into the memory space of the application:

```
glBindBuffer( buf, GL_ARRAY_BUFFER );
glBufferData( GL_ARRAY_BUFFER, 3*sizeof(float)*numVertices, NULL, GL_STATIC_DRAW );

float * vertexArray = glMapBuffer( GL_ARRAY_BUFFER,  usage );
```

Allocates the bytes, but
doesn't deliver any data

usage is an indication how the data will be used:

GL_READ_ONLY   the vertex data will be read from, but not written to
GL_WRITE_ONLY   the vertex data will be written to, but not read from
GL_READ_WRITE   the vertex data will be read from *and* written to

You can now use *vertexArray[ ]* like any other floating-point array.

When you are done, be sure to call:

```
glUnMapBuffer( GL_ARRAY_BUFFER );
```

Oregon State
University
Computer Graphics

mjb – March 27, 2021

We Now Pickup with the OpenCL Stuff

Oregon State
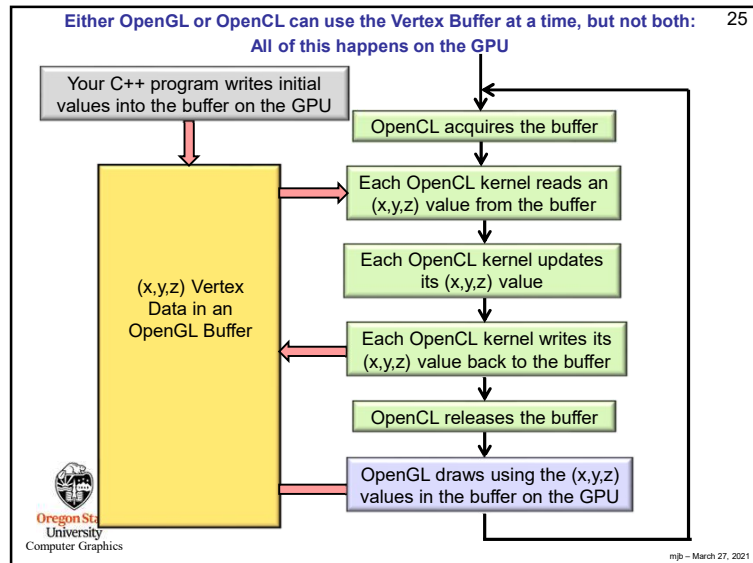University
Computer Graphics

mjb – March 27, 2021

We Now Pickup with the OpenCL Stuff

Oregon State
University
Computer Graphics

mjb – March 27, 2021

Slide 25

Either OpenGL or OpenCL can use the Vertex Buffer at a time, but not both:
All of this happens on the GPU

Your C++ program writes initial values into the buffer on the GPU

OpenCL acquires the buffer

Each OpenCL kernel reads an (x,y,z) value from the buffer

Each OpenCL kernel updates its (x,y,z) value

(x,y,z) Vertex Data in an OpenGL Buffer

Each OpenCL kernel writes its (x,y,z) value back to the buffer

OpenCL releases the buffer

OpenGL draws using the (x,y,z) values in the buffer on the GPU

Computer Graphics

mjb – March 27, 2021

25

Slide 26

1. Program Header

```
#include <stdio.h>
#define _USE_MATH_DEFINES
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <omp.h>


#ifdef WIN32
#include <windows.h>
#endif

#ifdef WIN32
#include "glew.h"
#endif

#include <GL/gl.h>
#include <GL/glu.h>
#include "glut.h"
#include "glui.h"

#include "CL/cl.h"
#include "CL/cl_gl.h"
```

Computer Graphics

mjb – March 27, 2021

26

Slide 27

Structures We Will Use to Fill the Vertex Buffers

```
// structs we will need later:

struct xyzw
{
        float x, y, z, w;
};


struct rgba
{
        float r, g, b, a;
};
```

Computer Graphics

mjb – March 27, 2021

27

Slide 28

OpenCL Global Variables

```
size_t  GlobalWorkSize[3]  = { NUM_PARTICLES, 1, 1 };
size_t  LocalWorkSize[3]   = { LOCAL_SIZE,    1, 1 };

GLuint                     hPobj;      // host opengl object for Points
GLuint                     hCobj;      // host opengl object for Colors
struct xyzw *              hVel;       // host C++ array for Velocities
cl_mem                     dPobj;      // device memory buffer for Points
cl_mem                     dCobj;      // device memory buffer for Colors
cl_mem                     dVel;       // device memory buffer for Velocities

cl_command_queue           CmdQueue;
cl_device_id               Device;
cl_kernel                  Kernel;
cl_platform_id             Platform;
cl_program                 Program;
```

Computer Graphics

mjb – March 27, 2021

28

A Deceptively-Simple Main Program

```
int
main( int argc, char *argv[ ] )
{
        glutInit( &argc, argv );
        InitGraphics(  );
        InitLists(  );
        InitCL(  );
        Reset(  );
        InitGlui(  );
        glutMainLoop(  );
        return 0;
}
```

Oregon State University
Computer Graphics

mjb – March 27, 2021

GLEW – the GL Extension Wrangler

```
#ifdef  WIN32
  GLenum err = glewInit( );
  if( err  !=  GLEW_OK )
  {
        fprintf( stderr, "glewInit Error\n" );
  }
#endif
```

This *must* wait to be called until after a graphics window is open !

Why? Because creating the window is what builds the graphics context.

Oregon State University
Computer Graphics

mjb – March 27, 2021

Setting up OpenCL:
Querying the Existence of an OpenCL Extension

```
void
InitCL( )
{
        . . .

        status = clGetDeviceIDs( Platform, CL_DEVICE_TYPE_GPU, 1, &Device, NULL );
        PrintCLError( status, "clGetDeviceIDs: " );

        // since this is an opengl interoperability program,
        // check if the opengl sharing extension is supported
        // (no point going on if it isn't):
        // (we need the Device in order to ask, so we can't do it any sooner than right here)

        if(  IsCLExtensionSupported( "cl_khr_gl_sharing" )  )
        {
                fprintf( stderr, "cl_khr_gl_sharing is supported.\n" );
        }
        else
        {
                fprintf( stderr, "cl_khr_gl_sharing is not supported -- sorry.\n" );
                return;
        }
```

Oregon State University
Computer Graphics

mjb – March 27, 2021

Querying the Existence of an OpenCL Extension

```
bool
IsCLExtensionSupported( const char *extension )
{
        // see if the extension is bogus:

        if( extension == NULL  ||  extension[0] == '\0' )
                return false;

        char * where = (char *) strchr( extension, ' ' );
        if( where != NULL )
                return false;

        // get the full list of extensions:

        size_t  extensionSize;
        clGetDeviceInfo( Device, CL_DEVICE_EXTENSIONS, 0, NULL, &extensionSize );
        char *extensions = new char [ extensionSize ];
        clGetDeviceInfo( Device, CL_DEVICE_EXTENSIONS, extensionSize, extensions, NULL );

        for( char * start = extensions ; ; )
        {
                where = (char *) strstr( (const char *) start, extension );
                if( where == 0 )
                {
                        delete [ ] extensions;
                        return false;
                }

                char * terminator = where + strlen(extension);  // points to what should be the separator

                if( *terminator == ' '  ||  *terminator == '\0'  ||  *terminator == '\r'  ||  *terminator == '\n' )
                {
                        delete [ ] extensions;
                        return true;
                }
                start = terminator;
        }
}
```

Oregon State University
Computer Graphics

mjb – March 27, 2021

Setting up OpenCL: The Interoperability Context

```
void
InitCL( )
{
        . . .

// get the platform id:

status = clGetPlatformIDs( 1, &Platform, NULL );
PrintCLError( status, "clGetPlatformIDs: " );

// get the device id:

status = clGetDeviceIDs( Platform, CL_DEVICE_TYPE_GPU, 1, &Device, NULL );
PrintCLError( status, "clGetDeviceIDs: " );

// 3. create a special opencl context based on the opengl context:

cl_context_properties props[ ] =
{
        CL_GL_CONTEXT_KHR,          (cl_context_properties) wglGetCurrentContext( ),
        CL_WGL_HDC_KHR,             (cl_context_properties) wglGetCurrentDC( ),
        CL_CONTEXT_PLATFORM,        (cl_context_properties) Platform,
        0
};

cl_context Context = clCreateContext( props, 1, &Device, NULL, NULL, &status );
PrintCLError( status, "clCreateContext: " );
```

University
Computer Graphics

mjb – March 27, 2021

33

Setting up OpenCL:
The Interoperability Context is Different for each OS (oh, good…)

```
For Windows:
cl_context_properties props[ ] =
{
        CL_GL_CONTEXT_KHR,          (cl_context_properties) wglGetCurrentContext( ),
        CL_WGL_HDC_KHR,             (cl_context_properties) wglGetCurrentDC( ),
        CL_CONTEXT_PLATFORM,        (cl_context_properties) Platform,
        0
};
cl_context Context = clCreateContext( props, 1, &Device, NULL, NULL, &status );

For Linux:
cl_context_properties props[ ] =
{
        CL_GL_CONTEXT_KHR,          (cl_context_properties) glXGetCurrentContext( ),
        CL_GLX_DISPLAY_KHR,         (cl_context_properties) glXGetCurrentDisplay( ),
        CL_CONTEXT_PLATFORM,        (cl_context_properties) Platform,
        0
};
cl_context Context = clCreateContext( props, 1, &Device, NULL, NULL, &status );

For Apple:
cl_context_properties props[ ] =
{
        CL_CONTEXT_PROPERTY_USE_CGL_SHAREGROUP_APPLE,
                                    (cl_context_properties) kCGLShareGroup,
        0
};
cl_context Context = clCreateContext( props, 0, 0, NULL, NULL, &status );
```

Computer Graphics

mjb – March 27, 2021

34

Setting up OpenCL

"hVel" stands for "host Velocities'"
"hPobj" stands for "host Points object"
"hCobj" stands for "host Colors object"

```
void
InitCL( )
{
        . . .

// create the velocity array and the opengl vertex array buffer and color array buffer:

delete [ ] hVel;
hVel = new struct xyzw [ NUM_PARTICLES ];

glGenBuffers( 1, &hPobj );
glBindBuffer( GL_ARRAY_BUFFER, hPobj );
glBufferData( GL_ARRAY_BUFFER, 4 * NUM_PARTICLES * sizeof(float), NULL, GL_STATIC_DRAW );

glGenBuffers( 1, &hCobj );
glBindBuffer( GL_ARRAY_BUFFER, hCobj );
glBufferData( GL_ARRAY_BUFFER, 4 * NUM_PARTICLES * sizeof(float), NULL, GL_STATIC_DRAW );

glBindBuffer( GL_ARRAY_BUFFER, 0 );        // unbind the buffer

// fill those arrays and buffers:

ResetParticles( );
```

Oregon State
University
Computer Graphics

mjb – March 27, 2021

35

Setting the Initial Particle Parameters, I

```
unsigned int Seed;
…
void
ResetParticles( )
{
        glBindBuffer( GL_ARRAY_BUFFER, hPobj );
        struct xyzw *points = (struct xyzw *) glMapBuffer( GL_ARRAY_BUFFER, GL_WRITE_ONLY );
        for( int i = 0; i < NUM_PARTICLES; i++ )
        {
                points[ i ].x = Ranf( &Seed, XMIN, XMAX );
                points[ i ].y = Ranf( &Seed, YMIN, YMAX );
                points[ i ].z = Ranf( &Seed, ZMIN, ZMAX );
                points[ i ].w = 1.;
        }
        glUnmapBuffer( GL_ARRAY_BUFFER );


        glBindBuffer( GL_ARRAY_BUFFER, hCobj );
        struct rgba *colors = (struct rgba *) glMapBuffer( GL_ARRAY_BUFFER, GL_WRITE_ONLY );
        for( int i = 0; i < NUM_PARTICLES; i++ )
        {
                colors[ i ].r = Ranf( &Seed, 0., 1. );
                colors[ i ].g = Ranf( &Seed, 0., 1. );
                colors[ i ].b = Ranf( &Seed, 0., 1. );
                colors[ i ].a = 1.;
        }
        glUnmapBuffer( GL_ARRAY_BUFFER );

        . . .
```

mjb – March 27, 2021

36

Setting the Initial Particle Parameters, II

```
. . .

for( int i = 0; i < NUM_PARTICLES; i++ )
{
          hVel[ i ].x = Ranf( &Seed, VMIN, VMAX );
          hVel[ i ].y = Ranf( &Seed, 0.      , VMAX );
          hVel[ i ].z = Ranf( &Seed, VMIN, VMAX );
          hVel[ i ].w = 0.;
}
}
```

Oregon State
University
Computer Graphics

mjb – March 27, 2021

37

Setting-up the Device-Side Buffers

```
void
InitCL( )
{
          . . .
```

// 5. create the opencl version of the velocity array:

dVel = **clCreateBuffer**(Context, CL_MEM_READ_WRITE, 4*sizeof(float)*NUM_PARTICLES, NULL, &status);
PrintCLError(status, "clCreateBuffer: ");

// 6. write the data from the host buffers to the device buffers:

status = **clEnqueueWriteBuffer**(CmdQueue, dVel, CL_FALSE, 0, 4*sizeof(float)*NUM_PARTICLES, hVel, 0, NULL, NULL);
PrintCLError(status, "clEneueueWriteBuffer: ");

// 5. create the opencl version of the opengl buffers:

dPobj = **clCreateFromGLBuffer**(Context, CL_MEM_READ_WRITE, **hPobj,** &status);
PrintCLError(status, "clCreateFromGLBuffer (1)");

dCobj = **clCreateFromGLBuffer**(Context, CL_MEM_READ_WRITE , **hCobj,** &status);
PrintCLError(status, "clCreateFromGLBuffer (2)");

Note: you don't need an OpenGL-accessible buffer for the velocities. Velocities aren't needed for drawing.
Velocities are only needed to update point positions. The velocity buffer can just be done internally to OpenCL.

University
Computer Graphics
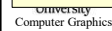
mjb – March 27, 2021

38

This is how OpenCL and OpenGL Share the Same Memory Buffer

dPobj = **clCreateFromGLBuffer**(Context, CL_MEM_READ_WRITE, **hPobj**, &status);
PrintCLError(status, "clCreateFromGLBuffer (1)");

Step #1: OpenGL creates the buffer on the GPU

Step #2: OpenCL is told about it and creates a device pointer to the already-filled
memory, just as if you had called **clCreateBuffer( )** and **clEnqueueWriteBuffer( )**

Oregon State
University
Computer Graphics

mjb – March 27, 2021

39

Setup the Kernel Arguments…

```
void
InitCL( )
{
          . . .
```

// 10. setup the arguments to the Kernel object:

status = clSetKernelArg(Kernel, **0**, sizeof(cl_mem), &dPobj);
PrintCLError(status,"clSetKernelArg (1): ");

status = clSetKernelArg(Kernel, **1**, sizeof(cl_mem), &dVel);
PrintCLError(status , "clSetKernelArg (2): ");

status = clSetKernelArg(Kernel, **2**, sizeof(cl_mem), &dCobj);
PrintCLError(status, "clSetKernelArg (3): ");

… to Match the Kernel's Parameter List

```
kernel
void
Particle( global point * dPobj,  global vector * dVel,  global color * dCobj )
{
          . . .
}
```

Oregon State
University
Computer Graphics

mjb – March 27, 2021

40

10

Slide 41

The OpenGL "Idle Function" Tells OpenCL to Do Its Computing 41

```
void
Animate( )
{
        // acquire the vertex buffers from opengl:

        glutSetWindow( MainWindow );
        glFinish( );

        cl_int status;
        status = clEnqueueAcquireGLObjects( CmdQueue, 1, &dPobj, 0, NULL, NULL );
        PrintCLError( status, "clEnqueueAcquireGLObjects (1) : " );
        status = clEnqueueAcquireGLObjects( CmdQueue, 1, &dCobj, 0, NULL, NULL );
        PrintCLError( status, "clEnqueueAcquireGLObjects (2) : " );

        Wait( ); // note: only need to wait here because doing timing
        double time0 = omp_get_wtime( );

        // 11. enqueue the Kernel object for execution:

        cl_event  wait;
        status = clEnqueueNDRangeKernel( CmdQueue, Kernel, 1, NULL, GlobalWorkSize, LocalWorkSize, 0, NULL, &wait );
        PrintCLError( status, "clEnqueueNDRangeKernel: " );

        Wait( );  // note: only need to wait here because doing timing
        double time1 = omp_get_wtime( );
        ElapsedTime  =  time1 - time0;

        clEnqueueReleaseGLObjects( CmdQueue, 1, &dCobj, 0, NULL, NULL );
        PrintCLError( status, "clEnqueueReleaseGLObjects (1) : " );
        clEnqueueReleaseGLObjects( CmdQueue, 1, &dPobj, 0, NULL, NULL );
        PrintCLError( status, "clEnqueueReleaseGLObject (2): " );

        Wait( );
        glutSetWindow( MainWindow );
        glutPostRedisplay( );
}
```

41

Slide 42

This is how OpenCL Manages Exclusive Access to the Memory Buffer 42

status = **clEnqueueAcquireGLObjects**(CmdQueue, 1, **&dPobj**, 0, NULL, NULL);

status = **clEnqueueAcquireGLObjects**(CmdQueue, 1, &dCobj, 0, NULL, NULL);

. . .

status = **clEnqueueReleaseGLObjects**(CmdQueue, 1, **&dCobj**, 0, NULL, NULL);

status = **clEnqueueReleaseGLObjects**(CmdQueue, 1, &dPobj, 0, NULL, NULL);

Oregon State
University
Computer Graphics

mjb – March 27, 2021

42

Slide 43

Redrawing the Scene:
The Particles 43

```
void
Display( )
{
        . . .

        glBindBuffer( GL_ARRAY_BUFFER, hPobj );
        glVertexPointer( 4, GL_FLOAT, 0, (void *)0 );
        glEnableClientState( GL_VERTEX_ARRAY );

        glBindBuffer( GL_ARRAY_BUFFER, hCobj );
        glColorPointer( 4, GL_FLOAT, 0, (void *)0 );
        glEnableClientState( GL_COLOR_ARRAY );

        glPointSize( 2. );
        glDrawArrays( GL_POINTS, 0, NUM_PARTICLES );
        glPointSize( 1. );

        glDisableClientState( GL_VERTEX_ARRAY );
        glDisableClientState( GL_COLOR_ARRAY );
        glBindBuffer( GL_ARRAY_BUFFER, 0 );

        glutSwapBuffers( );
        glFlush( );
}
```

Oregon State
University
Computer Graphics

mjb – March 27, 2021

43

Slide 44

Redraw the Scene:
The Performance 44

```
void
Display( )
{
        . . .

        if( ShowPerformance )
        {
                char str[128];
                sprintf( str, "%6.1f GigaParticles/Sec", (float)NUM_PARTICLES/ElapsedTime/1000000000. );
                glDisable( GL_DEPTH_TEST );
                glMatrixMode( GL_PROJECTION );
                glLoadIdentity( );
                gluOrtho2D( 0., 100.,    0., 100. );
                glMatrixMode( GL_MODELVIEW );
                glLoadIdentity( );
                glColor3f( 1., 1., 1. );
                DoRasterString( 5., 5., 0., str );
        }
}
```

1.2 GigaParticles/Sec

Oregon State
University
Computer Graphics

mjb – March 27, 2021

44

Slide 45

```
void
Quit( )
{
        Glui->close( );
        glutSetWindow( MainWindow );
        glFinish( );
        glutDestroyWindow( MainWindow );


        // 13. clean everything up:

        clReleaseKernel(            Kernel   );
        clReleaseProgram(           Program  );
        clReleaseCommandQueue(  CmdQueue );
        clReleaseMemObject(      dPobj  );
        clReleaseMemObject(      dCobj  );

        exit( 0 );
}
```

Oregon State University
Computer Graphics

mjb – March 27, 2021

45

Slide 46

```
typedef  float4  point;            // x, y, z – the w is unused
typedef  float4  vector;           // vx, vy, vz – the w is unused
typedef  float4  color;            // r, g, b – the w is unused
typedef  float4  sphere;           // xc, yc, zc, r

// despite what we think of the 4 components as representing,
// they are all referenced as .x, .y, .z, and .w

constant float4 G        = (float4) ( 0., -9.8, 0., 0. );        // gravity
constant float  DT       = 0.1;                                 // time step
constant sphere Sphere1 = (sphere)( -100., -800., 0., 600. );   // xc. yc, zc, r


bool
IsInsideSphere( point p, sphere s )
{
        float r = fast_length( p.xyz - s.xyz );
        return  ( r < s.w );
}
```

points, vectors, colors, and spheres are all represented as float4's. The typedefs help the program's readability by showing what that float4 is actually representing.

Oregon State University
Computer Graphics

mjb – March 27, 2021

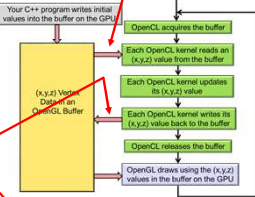46

Slide 47

```
kernel
void
Particle( global point * dPobj,  global vector * dVel,  global color * dCobj )
{
        int gid = get_global_id( 0 );      // particle #

        point p  = dPobj[gid];
        vector v = dVel[gid];

        point pp   = p + v*DT + G * (point)(.5*DT*DT); // p'
        vector vp = v + G*DT;                          // v'
        pp.w = 1.;
        vp.w = 0.;

        if(   IsInsideSphere( pp, Sphere1 )   )
        {
                vp = BounceSphere( p, v, Sphere1 );
                pp = p + vp*DT + G * (point)( .5*DT*DT );
        }

        dPobj[gid] = pp;
        dVel[gid]  = vp;
}
```

Your C++ program writes initial values into the buffer on the GPU

OpenCL acquires the buffer
Each OpenCL kernel reads an (x,y,z) value from the buffer
Each OpenCL kernel updates its (x,y,z) value
Each OpenCL kernel writes its (x,y,z) value back to the buffer
OpenCL releases the buffer
OpenGL draws using the (x,y,z) values in the buffer on the GPU

(x,y,z) Vector Data in an OpenGL Buffer

Computer Graphics Trick Alert: Making the bounce happen from the surface of the sphere is time-consuming to compute. Instead, bounce from the previous position in space. If DT is small enough, nobody will ever know…

Oregon State University
Computer Graphics

mjb – March 27, 2021

47

Slide 48
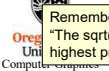
```
vector
Bounce( vector in,  vector n )
{
        n.w = 0.;
        n = fast_normalize( n );           // make it a unit vector

        // this is the vector equation for "angle of reflection equals angle of incidence":
        vector  out = in  -  n * (vector)( 2.*dot( in.xyz, n.xyz ) );
                // adding or subtracting 2 float4's gives you another float4
                // multiplying 2 float4's gives you another float4
                // when you want a dot product, use the dot( ) function
        out.w = 0.;
        return out;
}


vector
BounceSphere( point p, vector in,  sphere s )
{
        vector n;
        n.xyz = p.xyz - s.xyz;
                // the vector from the sphere center to the point is the normal
        return Bounce( in, n );
}
```
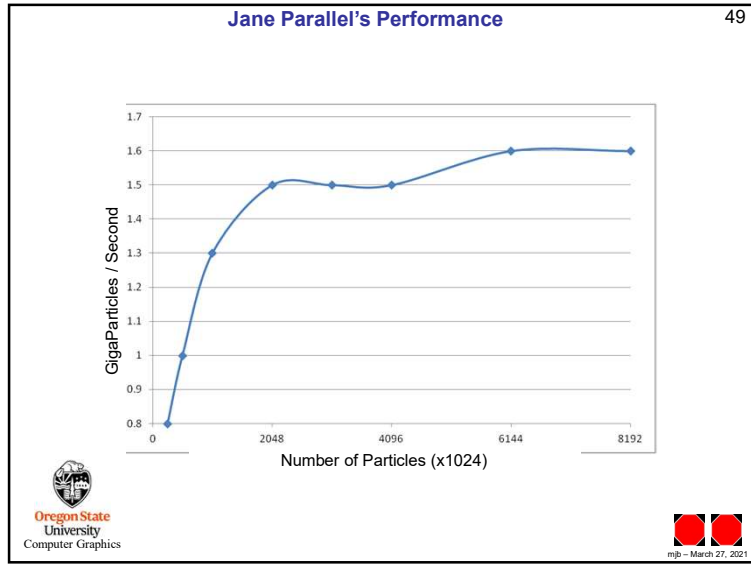
Remember from the OpenCL Assembly Language notes:
"The sqrt($x^2+y^2+z^2$) assembly code is amazingly involved. I suspect it is an issue of maintaining highest precision. Use **fast_sqrt( ), fast_normalize( ),** and **fast_length( )** when you can."

Oregon State University
Computer Graphics

mjb – March 27, 2021

48