Buffer Objects

Kurt Akeley, NVIDIA Corporation
Outline

- Background
- Buffer Objects
- Vertex Arrays
- Examples
Background

- **Geometry transfer is too slow**
  - Begin/End is inefficient
  - Vertex array memory management is poor
- **Vendor extensions are incompatible**
  - ATI_vertex_array_object
  - NV_vertex_array_range
  - Others
- **ATI and NVIDIA work together**
  - ARB_vertex_array_object
- **Result:** ARB_Vertex_Buffer_Object
Requirements

• High Performance
  – Optimize for static and dynamic data
  – Use the “best” memory
  – Provide mapped access

• Good Application Fit
  – Support mixed static/dynamic data sets
  – Support “mix and match” of vertex data
    • e.g. multiple tex coord arrays for one position array
    • e.g. constant color
  – Minimize code changes
Architecture

OpenGL defines the architecture for 3D visualization

Game Developers Conference

Make Better Games.
Architecture

Application → Memory + State → Buffer Object
Server-side state

- Allows sharing between GL contexts
- Matches use of GPU memory
- Good for GLX client/server rendering
Buffer Object

- Memory buffer
  - Array of basic machine units (bytes)
  - Data are in client format
- Small amount of state
  - Buffer size
  - Usage and access hints
  - Mapping state (Boolean and pointer)

Data format is implicit, not explicit
Basic API

```c
void GenBuffersARB(n, *buffers);
void BindBufferARB(target, buffer);
void DeleteBuffersARB(n, *buffers);
boolean IsBufferARB(buffer);
void GetBufferParameterivARB(target, pname, *params);
void GetBufferPointervARB(target, pname, **params);
```
Example

```c
uint buf;
int parameter;

GenBuffersARB(1, &buf);
BindBufferARB(GL_ARRAY_BUFFER_ARB, buf);
GetBufferParameterivARB(ARRAY_BUFFER_ARB,
                       BUFFER_SIZE_ARB, &parameter);
printf("Buffer size is %d\n", parameter);
DeleteBuffers(1, &buf);
```
Creating a Data Store

• New buffer objects have no data store
• `BufferDataARB(target, size, *data, usage)`
  – Discards any existing data store
  – Creates a new data store
  – Optionally initializes the contents
  – Specifies the intended usage pattern
• Usage hint discussed later
• Data alignment is per client requirements
• Re-initialization is inexpensive – do it
Changing Data Store Contents

• Two approaches
  – Functional interface (set and query)
  – Mapping

• Functional
  – BufferSubDataARB(target, offset, size, *data)
  – GetBufferSubDataARB(target, offset, size, *data)
  – This is the default approach
    • Static data
    • Array data
  – Always a safe approach
    • Data are never corrupted
Mapping a Buffer Object

- Intended for data streams
- `void *MapBufferARB(target, access)`
  - `READ_ONLY_ARB`, `WRITE_ONLY_ARB`, `READ_WRITE_ARB`
  - Maps the entire data store
  - Returns a pointer to the buffer memory
  - May be slow if data are copied
  - May result in data loss
- `boolean UnmapBufferARB(target)`
  - Returns true if data are uncorrupted
  - Invalidates pointer
Mapping Rules

- **Specify the correct `access` value**
  - Otherwise operation is undefined

- **Be prepared for data loss**
  - Use functional interface if this is a burden

- **Don’t render from a mapped buffer**
  - The error `INVALID_OPERATION` results

- **Map for brief periods only**
  - Map it, modify it, then unmap it

- **Don’t pass a map pointer to the GL**
Summary

• Buffer objects
  – Unformatted, server-side memory buffers
  – Include a small amount of state

• Two ways to modify buffer contents
  – Functional interface
  – Direct mapping

• Very general mechanism
  – Could work for any GL data stream
  – Implemented for vertex arrays
Vertex Arrays

- Vertex arrays are application state
- Two-step process
  - App specifies array locations and formats
  - GL pulls vertex data from arrays
- Goals
  - Store vertex arrays in buffer objects
  - Maximize flexibility
  - Avoid misuse of the mapping pointer
  - Avoid a cluttered, incompatible API
Per-Array Buffer Specification

VertexPointer(size, type, stride, *pointer);

ARRAY_BUFFER_ARB

offset

size

pointer

type

stride

buffer

Game Developers Conference

Make Better Games.
Client and Server State

- Buffer objects are server state
- Vertex arrays parameters are client state
Usage Terms

• **Stream**
  - Specify once
  - Render once

• **Static**
  - Specify once
  - Render repeatedly

• **Dynamic**
  - Everything else
  - Specify/modify repeatedly
  - Render repeatedly
Usage Terms

Application + State → Memory → Copy

Read

Draw

OpenGL defines the architecture for 3D visualization

Game Developers Conference

Make Better Games
## Usages

<table>
<thead>
<tr>
<th></th>
<th>Draw</th>
<th>Read</th>
<th>Copy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Stream</strong></td>
<td>STREAM_DRAW</td>
<td>STREAM_READ</td>
<td>STREAM_COPY</td>
</tr>
<tr>
<td><strong>Static</strong></td>
<td>STATIC_DRAW</td>
<td>STATIC_READ</td>
<td>STATIC_COPY</td>
</tr>
<tr>
<td><strong>Dynamic</strong></td>
<td>DYNAMIC_DRAW</td>
<td>DYNAMIC_READ</td>
<td>DYNAMIC_COPY</td>
</tr>
</tbody>
</table>

**Vertex Array Usage**

---

*Image and text are copyrighted.*
Example

```
#define BUFFER_OFFSET(i) ((char *)NULL + (i))

data = malloc(320);
...

// Fill system memory buffer
BindBufferARB(ARRAY_BUFFER_ARB, 1);
BufferDataARB(ARRAY_BUFFER_ARB, 320, data, STATIC_DRAW_ARB);
free(data);
while (...) {
    BindBufferARB(ARRAY_BUFFER_ARB, 1); // must precede pointer cmds
    VertexPointer(4, FLOAT, 0, BUFFER_OFFSET(0));
    ColorPointer(4, UNSIGNED_BYTE, 0, BUFFER_OFFSET(256));
    EnableClientState(VERTEX_ARRAY);
    EnableClientState(COLOR_ARRAY);
    DrawArrays(TRIANGLE_STRIP, 0, 16);
...
}
```

Notes

• Index arrays are supported
  - ELEMENT_ARRAY_BUFFER_ARB

• Other extensions are supported
  - EXT_vertex_shader
  - ARB_vertex_program
  - ...

• Display lists are not supported
• intptr and sizeofptr types are introduced
• GLX protocol is not yet defined
Tips

• Keep static and dynamic data in separate buffer objects
• Keep vertex and index data separate
• Bind to the “correct” target
• Reinitialize data buffers
• Use mapping carefully
  – stream data
  – volatile memory
• More extensions coming soon (PBO)