OpenGL extension for the G8x

NV_parameter_buffer_object & NV_transform_feedback

NV_parameter_buffer_object

- Overview
  - New type program parameter => buffer object (array)
  - Each program(vertex, fragment, geometry) has a set of binding points to which buffer objects can be attached
    - program.buffer [a] [b]
    - a => binding point of the program
    - program.buffer [a] => buffer object that is bound to point ‘a’
    - b => buffer object array index
  - Change large blocks of program parameter at once, by binding a new buffer object

- Dependency
  - OpenGL 2.0 & NV_gpu_program4 are required

Buffer object functions

- void GenBuffers(sizei n, uint *buffers);
  - Generate ‘n’ previously unused buffer object in ‘buffers’ array

- void BindBuffer(enum target, uint buffer);
  - Bind ‘buffer’ to ‘target’ (‘buffer’ is generated by GetBuffers())
  - ‘target’ must be one of ARRAY_BUFFER , ELEMENT_ARRAY_BUFFER , PIXEL_UNPACK_BUFFER , PIXEL_PACK_BUFFER

- void BufferData(enum target, sizeiptr size, const void *data, enum usage);
  - Create and initialize memory space of buffer object
  - ‘target’ is the same as BindBuffer()’s ‘target’
  - ‘size’ is the size of memory space
  - ‘data’ is copied to buffer object’s memory space. ‘data’ can be NULL
  - one of nine enumerated values
    - STATIC_DRAW, STATIC_READ, STATIC_COPY, DYNAMIC_ … , STREAM_ …

- void BufferSubData(enum target, intptr offset, sizeiptr size, const void *data);
  - Copy data to buffer object
  - 'target' is the same as BindBuffer()'s 'target'
  - 'offset' and 'size' indicate the range of data in the buffer object that is to be replaced
  - 'data' contains memory to copy

- void *MapBuffer(enum target, enum access);
  - Buffer object's memory address maps into applications memory address
  - 'access' is a flag (READ_WRITE, READ_ONLY, WRITE_ONLY)

- boolean UnmapBuffer(enum target);
  - Unmap mapped buffer
- void DeleteBuffers(sizei n, const uint *buffers);
  - Delete 'n' buffer object in 'buffers' array

New functions

- void BindBufferBaseNV(enum target, uint index, uint buffer);
  - ‘target’ is one of three enum (program to bind)
    - VERTEX_PROGRAM_PARAMETER_BUFFER_NV
    - GEOMETRY_PROGRAM_PARAMETER_BUFFER_NV
    - FRAGMENT_PROGRAM_PARAMETER_BUFFER_NV
  - ‘index’ indicates binding point of the program
  - ‘buffer’ indicates buffer object to bind

- void BindBufferOffsetNV(enum target, uint index, uint buffer, intptr offset);
  - ‘offset’ is a starting offset into the buffer object

- void BindBufferRangeNV(enum target, uint index, uint buffer, intptr offset, sizeiptr size);
  - ‘size’ specifies the number of elements to bind in buffer

- void ProgramBufferParametersfvNV(enum target, uint buffer, uint index, sizei count, const float *params);
  - Update memory of buffer object
  - ‘target’ is the same
  - ‘buffer’ is the buffer object
  - ‘index’ => starting point of buffer object to update
  - ‘count’ => number of element to update
  - ‘params’ contains data to update into buffer object

- void ProgramBufferParametersIivNV(enum target, uint buffer, uint index, sizei count, const int *params);
  - Same function for integer data

- void ProgramBufferParametersIuivNV(enum target, uint buffer, uint index, sizei count, const uint *params);
  - Same function for unsigned integer data

Example

Application Program

```
GLfloat buffer[64] ;
for (int i = 0; i < 64; ++i) {
                buffer[i] = 0 ;
}
GLuint bi ;

glGenBuffers(1, &bi) ;
glBindBuffer(GL_ARRAY_BUFFER, bi);

// set buffer size
glBufferData(GL_ARRAY_BUFFER, sizeof(GLfloat )*16, NULL, GL_STATIC_DRAW);
// binding point 7
glBindBufferBaseNV(GL_VERTEX_PROGRAM_PARAMETER_BUFFER_NV, 7, bi) ;
// update the content of buffer object with client 'buffer'
glProgramBufferParametersfvNV(GL_VERTEX_PROGRAM_PARAMETER_BUFFER_NV,
                7, 0, 16, buffer) ;
// mapping
float *tt = (float *)glMapBuffer(GL_ARRAY_BUFFER, GL_READ_WRITE) ;
tt[9] = 0.8 ;  // update element at index[9]

glUnmapBuffer(GL_ARRAY_BUFFER) ;
```

Example

Shader Program

```
!!NVvp4.0
ATTRIB iPos        = vertex.position;
ATTRIB iColor     = vertex.color;
PARAM  mvp[4]   = { state.matrix.mvp };
OUTPUT oPos       = result.position;
OUTPUT oColor   = result.color;
PARAM cc = {0.0, 0.0, 0.0, 1.0} ;

BUFFER tb[] = { program.buffer[7] };

DP4 oPos.x, mvp[0], iPos;
DP4 oPos.y, mvp[1], iPos;
DP4 oPos.z, mvp[2], iPos;
DP4 oPos.w, mvp[3], iPos;

MOV oColor, cc;
MOV oColor.r, tb[9];

END
```

- Size of buffer object => 16
- Binding point => 7
- read buffer element at index [9] ('0.8' is stored)
- and put the value into 'r' color

Issues

- Buffer object parameter is environment parameter
- Use new functions instead of ‘ProgramEnvParameter API’
- Reference to the parameter buffer is ‘scalar’
- parameter buffers is editable outside the ‘ProgramBufferParameters API’
- There is a limit to the size of program parameter buffer

NV_transform_feedback

- Overview
  - Records vertex attributes of the primitives processed by GL
    - Transform feedback occurs before clipping in the middle of pipeline
  - Selected attributes are written into buffer object
    - All attributes are written into one buffer object in interleaved form
    - Each attribute is written into a separate buffer object
  - Query object
    - determine when transform feedback results are complete
    - number of primitives processed (before clipping in the pipeline)
    - number of primitives written back to buffer object
    - Begin ~ End function structure
  - Provide rasterizer discard mode

- Dependency
  - OpenGL 1.5 is required
  - ARB_shader_objects , OpenGL 2.0
    - When we use assembly shader program
    - => we can't use some stuff like (TransformFeedbackVaryingsNV, GetVaryingLocationNV, GetActiveVaryingNV, ActiveVaryingNV, and GetTransformFeedbackVaryingNV functions and relevant enum)
  - NV_vertex_program4 , NV_geometry_program4
    - If It's not supported, we can't use PRIMITIVE_ID, VERTEX_ID stuff

Example

Application Program (initializing stuff)

```
 GLuint tf_bi1, tf_bi2 ;  // buffer object 1, 2
GLuint q, q2 ;            // query object 1, 2

// generate buffer object 1
glGenBuffers(1, &tf_bi1) ;
// bind
glBindBuffer(GL_ARRAY_BUFFER, tf_bi1);
// set size
glBufferData(GL_ARRAY_BUFFER, sizeof(float)*64, NULL, GL_STATIC_DRAW);

glGenBuffers(1, &tf_bi2) ;
glBindBuffer(GL_ARRAY_BUFFER, tf_bi2);
glBufferData(GL_ARRAY_BUFFER, sizeof(float)*64, NULL, GL_STATIC_DRAW);

// generate query object 1
glGenQueries(1, &q);
glGenQueries(1, &q2);
```

Example

Application Program (using tranform_feedback)   part 1

```
glBindBufferBaseNV(GL_TRANSFORM_FEEDBACK_BUFFER_NV, 0, tf_bi1);
glBindBufferBaseNV(GL_TRANSFORM_FEEDBACK_BUFFER_NV, 1, tf_bi2);

int attr[6] ;
attr[0] = GL_POSITION ;
attr[1] = 4 ;
attr[2] = 0 ;
attr[3] = GL_PRIMARY_COLOR ;
attr[4] = 4 ;
attr[5] = 0 ;
glTransformFeedbackAttribsNV(2,attr, GL_SEPARATE_ATTRIBS_NV) ;

glBeginTransformFeedbackNV(GL_TRIANGLES);
glBeginQuery(GL_PRIMITIVES_GENERATED_NV, q);
glBeginQuery(GL_TRANSFORM_FEEDBACK_PRIMITIVES_WRITTEN_NV, q2);

        glBegin(GL_TRIANGLES);
                glColor3f(1.0f,0.0f,0.0f);
                glVertex3f( 0.0f, 1.0f, 0.0f);
                        ….
        glEnd();
```

Functions (1/4)

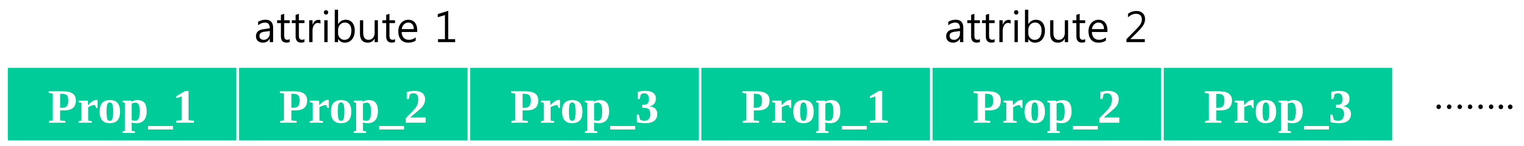
- Bind function
  - void BindBufferBaseNV(enum target, uint index, uint buffer);
  - void BindBufferOffsetNV(enum target, uint index, uint buffer, intptr offset);
  - void BindBufferRangeNV(enum target, uint index, uint buffer, intptr offset, sizeiptr size);

  - for 'target', set TRANSFORM_FEEDBACK_BUFFER_NV
  - for 'index'
    - Interleaved mode => index is '0'
    - Separate mode => index starts from '0' to 'n-1' (n : number of buffer object)

- void TransformFeedbackAttribsNV(sizei count, const int *attribs, enum bufferMode);
  - Choose attributes to write to buffer object

Functions (2/4)

- 'bufferMode'
  - SEPARATE_ATTRIBS_NV => write each attribute to separate buffer object
  - INTERLEAVED_ATTRIBS_NV => write all attributes to one buffer object
- 'count' => number of attributes to write
- 'attribs'
  - Array that contains each attribute's properties
  - For one attribute, it stores 3 property value (3*i+0, 3*i+1, 3*i+2)



  - property 1 => sort of attribute (like position, color)
  - property 2 => attribute size (for example, 'position' has 4 kinds of size
    1 : (x), 2 : (x, y), 3 : (x, y, z), 4: (x, y, z, w))
  - property 3 => index of attribute to write (if the attribute is vector form, like
    texture_coord[]).
  - See the followed table

Functions (3/4)

attribute	permitted size	index?	GPU_program_4 result name
POSITION	1,2,3,4	no	position
PRIMARY_COLOR	1,2,3,4	no	color.front.primary
SECONDARY_COLOR_NV	1,2,3,4	no	color.front.secondary
BACK_PRIMARY_COLOR_NV	1,2,3,4	no	color.back.primary
BACK_SECONDARY_COLOR_NV	1,2,3,4	no	color.back.secondary
FOG_COORDINATE	1	no	fogcoord
POINT_SIZE	1	no	pointsize
TEXTURE_COORD_NV	1,2,3,4	yes	texcoord[index]
CLIP_DISTANCE_NV	1	yes	clip[index]
VERTEX_ID_NV	1	no	vertexid
PRIMITIVE_ID_NV	1	no	primid
GENERIC_ATTRIB_NV	1,2,3,4	yes	attrib[index]

Function (4/4)

- void BeginTransformFeedbackNV(enum primitiveMode);
  - Set active mode for transform_feedback
  - When the mode is active, the primitives processed by GL are written to buffer
  - 'primitiveMode' is one of (POINTS, LINES, TRIANGLES)
  - There is corresponding between 'primitiveMode' and 'render primitive mode'
    - POINTS : POINTS
    - LINES : LINES, LINE_LOOP, and LINE_STRIP
    - TRIANGLES : TRIANGLES, TRIANGLE_STRIP, TRIANGLE_FAN, QUADS, QUAD_STRIP, and POLYGON

- void BeginQuery(enum target, uint id);
  - Create 'id' query object
  - 'target'
    - PRIMITIVES_GENERATED_NV => records the number of primitives processed by GL
    - TRANSFORM_FEEDBACK_PRIMITIVES_WRITTEN_NV => records the number of primitives that are written to buffer object

Example

Application Program (using tranform_feedback)   part 2

```
// end query
glEndQuery(GL_PRIMITIVES_GENERATED_NV);
glEndQuery(GL_TRANSFORM_FEEDBACK_PRIMITIVES_WRITTEN_NV);

// set inactive mode for transform_feedback
glEndTransformFeedbackNV();

// get the number of primitives processed by GL
int gen=-1;
glGetQueryObjectiv(q, GL_QUERY_RESULT, &gen);

// get the number of primitives written to buffer object
int written = -1 ;
glGetQueryObjectiv(q2, GL_QUERY_RESULT, &written);
```

- void GetQueryObjectiv(uint id, enum pname, int *params);
  - Get state from ‘id’ query object
  - ‘pname’ => if QUERY_RESULT, returns query’s result value
  - ‘params’ => variable to store return value

Example

Application Program (using tranform_feedback)   part 3

```
 if (gen != written) {
          // if error occurs
          printf("number of written vertices is different\n")  ;
}
else {
          // print the contents of buffer object  (  position is written)
          glBindBuffer(GL_ARRAY_BUFFER, tf_bi1);
          float *tt = (float *)glMapBuffer(GL_ARRAY_BUFFER, GL_READ_WRITE) ;
          for (int i = 0; i < 20; ++i) {
                    printf("%f ", tt[i]) ;      // for test
          }
          glUnmapBuffer(GL_ARRAY_BUFFER) ;

          // print the contents of buffer object  (  color is written)
          glBindBuffer(GL_ARRAY_BUFFER, tf_bi2);
          tt = (float *)glMapBuffer(GL_ARRAY_BUFFER, GL_READ_WRITE) ;
          for (int i = 0; i < 20; ++i) {
                    printf("%f ", tt[i]) ;      // for test
          }
          glUnmapBuffer(GL_ARRAY_BUFFER) ;

}
```

Issues

- Provides rasterizer discard mode
  - glEnable(GL_RASTERIZER_DISCARD_NV) ;

- Expected usage enum for glBufferData() in conjunction with feedback
  - STREAM_COPY , STREAM_READ are common
  - Buffer object is written by GL and subsequently consumed by GL

    => _COPY
  - Buffer object is written by GL and consumed by App

    => _READ

- For incomplete primitives
  - none of the vertices in that primitive are written to the buffer