Компьютерная графика

Лекция 3: Объекты OpenGL, буферы, аттрибуты вершин, перспективная проекция

2021

Объекты OpenGL

- Шейдеры, шейдерные программы - программируемая часть конвейера

Объекты OpenGL

- Шейдеры, шейдерные программы - программируемая часть конвейера
- Буферы - хранят данные на GPU

Объекты OpenGL

- Шейдеры, шейдерные программы - программируемая часть конвейера
- Буферы - хранят данные на GPU
- Vertex Array - описывают аттрибуты вершин

Объекты OpenGL

- Шейдеры, шейдерные программы - программируемая часть конвейера
- Буферы - хранят данные на GPU
- Vertex Array - описывают аттрибуты вершин
- Текстуры - изображения, которые можно читать из шейдера, и в которые можно рисовать

Объекты OpenGL

- Шейдеры, шейдерные программы - программируемая часть конвейера
- Буферы - хранят данные на GPU
- Vertex Array - описывают аттрибуты вершин
- Текстуры - изображения, которые можно читать из шейдера, и в которые можно рисовать
- Renderbuffer - буферы, в которые можно рисовать

Объекты OpenGL

- Шейдеры, шейдерные программы - программируемая часть конвейера
- Буферы - хранят данные на GPU
- Vertex Array - описывают аттрибуты вершин
- Текстуры - изображения, которые можно читать из шейдера, и в которые можно рисовать
- Renderbuffer - буферы, в которые можно рисовать
- Framebuffer - содержат настройки рисования в текстуры и renderbuffer'ы

Создание/удаление объектов OpenGL

- Объект представляется идентификатором типа `GLuint`
  - Id уникален среди объектов одного типа (шейдеры, программы, ...)

Создание/удаление объектов OpenGL

- Объект представляется идентификатором типа `GLuint`
  - Id уникален среди объектов одного типа (шейдеры, программы, ...)
- Шейдеры и программы:
  - `glCreateShader()`/`glDeleteShader(shader)`
  - `glCreateProgram()`/`glDeleteProgram(program)`

Создание/удаление объектов OpenGL

- Объект представляется идентификатором типа `GLuint`
  - Id уникален среди объектов одного типа (шейдеры, программы, ...)
- Шейдеры и программы:
  - `glCreateShader()/glDeleteShader(shader)`
  - `glCreateProgram()/glDeleteProgram(program)`
- Остальные объекты:

`glGenBuffers(count, ptr)/glDeleteBuffers(count, ptr)`
`glGenVertexArrays/glDeleteVertexArrays`
`glGenTextures/glDeleteTextures`
`glGenRenderbuffers/glDeleteRenderbuffers`
`glGenFramebuffers/glDeleteFramebuffers`

Создание/удаление объектов OpenGL

- Объект представляется идентификатором типа `GLuint`
  - Id уникален среди объектов одного типа (шейдеры, программы, ...)
- Шейдеры и программы:
  - `glCreateShader()/glDeleteShader(shader)`
  - `glCreateProgram()/glDeleteProgram(program)`
- Остальные объекты:

  `glGenBuffers(count, ptr)/glDeleteBuffers(count, ptr)`
  `glGenVertexArrays/glDeleteVertexArrays`
  `glGenTextures/glDeleteTextures`
  `glGenRenderbuffers/glDeleteRenderbuffers`
  `glGenFramebuffers/glDeleteFramebuffers`

  - Можно создать/удалить один объект:

```
GLuint texture;
glGenTextures(1, &texture);
...
glDeleteTexture(1, &texture);
```

Объекты OpenGL

- Подразумевается, что объекты переиспользуются по максимуму
- Не нужно создавать новую текстуру каждый кадр - создайте один раз и переиспользуйте её

Объекты OpenGL с нулевым id

- Как правило, объект с нулевым id считается несуществующим (как нулевой указатель)

Объекты OpenGL с нулевым id

- Как правило, объект с нулевым id считается несуществующим (как нулевой указатель)
- Исключения:
  - Framebuffer с нулевым id - default framebuffer, рисует в окно, привязанное к контексту OpenGL

Объекты OpenGL с нулевым id

- Как правило, объект с нулевым id считается несуществующим (как нулевой указатель)
- Исключения:
  - Framebuffer с нулевым id - default framebuffer, рисует в окно, привязанное к контексту OpenGL
  - В OpenGL ES: Vertex Array с нулевым id - ничем не отличается от других vertex array'ев, но существует (создан) по умолчанию

Объекты OpenGL с нулевым id

- Как правило, объект с нулевым id считается несуществующим (как нулевой указатель)
- Исключения:
  - Framebuffer с нулевым id - default framebuffer, рисует в окно, привязанное к контексту OpenGL
  - В OpenGL ES: Vertex Array с нулевым id - ничем не отличается от других vertex array'ев, но существует (создан) по умолчанию
- Функции создания объектов никогда не возвращают нулевой id

Работа с объектами OpenGL

- Почти всегда чтобы работать с объектом, нужно сделать его "текущим"
  - Текущий объект запоминается в контексте OpenGL
  - Если вы работаете с одним контекстом, можно считать, что id текущего объекта - глобальная константа

Работа с объектами OpenGL

- Почти всегда чтобы работать с объектом, нужно сделать его "текущим"
  - Текущий объект запоминается в контексте OpenGL
  - Если вы работаете с одним контекстом, можно считать, что id текущего объекта - глобальная константа
- Некоторые функции не требуют выставления объекта текущим: `glShaderSource`, `glCompileShader`, `glLinkProgram`, ...

Работа с объектами OpenGL

- Почти всегда чтобы работать с объектом, нужно сделать его "текущим"
  - Текущий объект запоминается в контексте OpenGL
  - Если вы работаете с одним контекстом, можно считать, что id текущего объекта - глобальная константа
- Некоторые функции не требуют выставления объекта текущим: `glShaderSource`, `glCompileShader`, `glLinkProgram`, ...
- Некоторые объекты нельзя сделать текущими - шейдеры

Работа с объектами OpenGL

- Сделать программу текущей: `glUseProgram(program)`
  - Функции `glGetUniformLocation`, `glUniform1f`, ... работают с текущей программой

Работа с объектами OpenGL

- Сделать программу текущей: `glUseProgram(program)`
  - Функции `glGetUniformLocation`, `glUniform1f`, ... работают с текущей программой
- Сделать vertex array текущим: `glBindVertexArray(vao)`
  - Функции работы с vertex array используют текущий vertex array

Работа с объектами OpenGL

- Сделать программу текущей: `glUseProgram(program)`
  - Функции `glGetUniformLocation`, `glUniform1f`, ... работают с текущей программой
- Сделать vertex array текущим: `glBindVertexArray(vao)`
  - Функции работы с vertex array используют текущий vertex array
- Функции рисования (`glDrawArrays`) используют текущую шейдерную программу и текущий vertex array

Работа с объектами OpenGL

- Для буферов, текстур, framebuffer'ов и renderbuffer'ов нет одного текущего объекта, но есть текущий объект для конкретного target'а

Работа с объектами OpenGL

- Для буферов, текстур, framebuffer'ов и renderbuffer'ов нет одного текущего объекта, но есть текущий объект для конкретного target'а
- Можно считать, что есть словарь `Target -> Id` текущих объектов

Работа с объектами OpenGL

- Для буферов, текстур, framebuffer'ов и renderbuffer'ов нет одного текущего объекта, но есть текущий объект для конкретного target'а
- Можно считать, что есть словарь `Target -> Id` текущих объектов
- Для каждого вида объектов (буферы, текстуры, ...) есть отдельный словарь и отдельный набор возможных значений Target

Работа с объектами OpenGL

- Для буферов, текстур, framebuffer'ов и renderbuffer'ов нет одного текущего объекта, но есть текущий объект для конкретного target'а
- Можно считать, что есть словарь `Target -> Id` текущих объектов
- Для каждого вида объектов (буферы, текстуры, ...) есть отдельный словарь и отдельный набор возможных значений Target
- Смысл и особенности разных значений Target зависят от вида объекта

Работа с объектами OpenGL

- Для буферов, текстур, framebuffer'ов и renderbuffer'ов нет одного текущего объекта, но есть текущий объект для конкретного target'а
- Можно считать, что есть словарь `Target -> Id` текущих объектов
- Для каждого вида объектов (буферы, текстуры, ...) есть отдельный словарь и отдельный набор возможных значений Target
- Смысл и особенности разных значений Target зависят от вида объекта
- `glBindBuffer(target, id)`
- `glBindTexture(target, id)`
- `glBindRenderbuffer(target, id)`
- `glBindFramebuffer(target, id)`

Буферы

- Могут хранить произвольные данные на GPU

Буферы

- Могут хранить произвольные данные на GPU
- `glGenBuffers/glDeleteBuffers`

Буферы

- Могут хранить произвольные данные на GPU
- `glGenBuffers`/`glDeleteBuffers`
- Возможные значения target для `glBindBuffer`:
  - `GL_ARRAY_BUFFER` (VBO) - массив вершин

Буферы

- Могут хранить произвольные данные на GPU
- `glGenBuffers`/`glDeleteBuffers`
- Возможные значения target для `glBindBuffer`:
  - `GL_ARRAY_BUFFER` (VBO) - массив вершин
  - `GL_ELEMENT_ARRAY_BUFFER` (EBO) - массив индексов вершин

Буферы

- Могут хранить произвольные данные на GPU
- `glGenBuffers`/`glDeleteBuffers`
- Возможные значения target для `glBindBuffer`:
  - `GL_ARRAY_BUFFER` (VBO) - массив вершин
  - `GL_ELEMENT_ARRAY_BUFFER` (EBO) - массив индексов вершин
  - `GL_UNIFORM_BUFFER` (UBO) - массив значений uniform-переменных

Буферы

- Могут хранить произвольные данные на GPU
- `glGenBuffers`/`glDeleteBuffers`
- Возможные значения target для `glBindBuffer`:
  - `GL_ARRAY_BUFFER` (VBO) - массив вершин
  - `GL_ELEMENT_ARRAY_BUFFER` (EBO) - массив индексов вершин
  - `GL_UNIFORM_BUFFER` (UBO) - массив значений uniform-переменных
  - ...и другие

Буферы

- Могут хранить произвольные данные на GPU
- `glGenBuffers`/`glDeleteBuffers`
- Возможные значения target для `glBindBuffer`:
  - `GL_ARRAY_BUFFER` (VBO) - массив вершин
  - `GL_ELEMENT_ARRAY_BUFFER` (EBO) - массив индексов вершин
  - `GL_UNIFORM_BUFFER` (UBO) - массив значений uniform-переменных
  - ...и другие
- Текущий `GL_ELEMENT_ARRAY_BUFFER` хранится не глобально, а в текущем VAO

Буферы: запись данных

- Загрузить данные в буфер:

```
glBufferData(GLenum target, GLsizeiptr size,
    const GLvoid * data, GLenum usage)
```

Буферы: запись данных

- Загрузить данные в буфер:

```
glBufferData(GLenum target, GLsizeiptr size,
    const GLvoid * data, GLenum usage)
```

  - `target` - `GL_ARRAY_BUFFER` и т.п.
  - `size` - размер данных в байтах
  - `data` - указатель на данные
  - `usage` - подсказка драйверу о том, как данные будут использоваться

Буферы: запись данных

- Загрузить данные в буфер:

```
glBufferData(GLenum target, GLsizeiptr size,
    const GLvoid * data, GLenum usage)
```

  - `target` - `GL_ARRAY_BUFFER` и т.п.
  - `size` - размер данных в байтах
  - `data` - указатель на данные
  - `usage` - подсказка драйверу о том, как данные будут использоваться
- Если `data = nullptr`, буфер будет выделен, но данные будут не инициализированы

Буферы: запись данных

- Загрузить данные в буфер:

```
glBufferData(GLenum target, GLsizeiptr size,
    const GLvoid * data, GLenum usage)
```

  - `target` - `GL_ARRAY_BUFFER` и т.п.
  - `size` - размер данных в байтах
  - `data` - указатель на данные
  - `usage` - подсказка драйверу о том, как данные будут использоваться
- Если `data = nullptr`, буфер будет выделен, но данные будут не инициализированы
- Если буфер уже содержал данные, они заменяются новыми (и происходит реаллокация памяти)

Буферы: запись данных

- Загрузить данные в буфер:

```
glBufferData(GLenum target, GLsizeiptr size,
    const GLvoid * data, GLenum usage)
```

  - `target` - `GL_ARRAY_BUFFER` и т.п.
  - `size` - размер данных в байтах
  - `data` - указатель на данные
  - `usage` - подсказка драйверу о том, как данные будут использоваться
- Если `data = nullptr`, буфер будет выделен, но данные будут не инициализированы
- Если буфер уже содержал данные, они заменяются новыми (и происходит реаллокация памяти)
- После вызова `glBufferData` с данными по указателю `data` можно делать всё, что угодно (в т.ч. удалить)
- Копирование данных в память GPU тоже происходит асинхронно

Буферы: запись данных

- Загрузить данные в буфер:

```
glBufferData(GLenum target, GLsizeiptr size,
    const GLvoid * data, GLenum usage)
```

  - `target` - `GL_ARRAY_BUFFER` и т.п.
  - `size` - размер данных в байтах
  - `data` - указатель на данные
  - `usage` - подсказка драйверу о том, как данные будут использоваться
- Если `data = nullptr`, буфер будет выделен, но данные будут не инициализированы
- Если буфер уже содержал данные, они заменяются новыми (и происходит реаллокация памяти)
- После вызова `glBufferData` с данными по указателю `data` можно делать всё, что угодно (в т.ч. удалить)
- Копирование данных в память GPU тоже происходит асинхронно
  - $\Rightarrow$ Драйвер, скорее всего, сначала копирует данные в собственную память

Буферы: параметр usage

GL_STATIC_DRAW	GL_STATIC_READ	GL_STATIC_COPY
GL_DYNAMIC_DRAW	GL_DYNAMIC_READ	GL_DYNAMIC_COPY
GL_STREAM_DRAW	GL_STREAM_READ	GL_STREAM_COPY

Буферы: параметр usage

`GL_STATIC_DRAW`	`GL_STATIC_READ`	`GL_STATIC_COPY`
`GL_DYNAMIC_DRAW`	`GL_DYNAMIC_READ`	`GL_DYNAMIC_COPY`
`GL_STREAM_DRAW`	`GL_STREAM_READ`	`GL_STREAM_COPY`

- Данные будут обновляться
  - `STATIC` - один раз
  - `DYNAMIC` - иногда
  - `STREAM` - почти каждый кадр

Буферы: параметр usage

`GL_STATIC_DRAW`	`GL_STATIC_READ`	`GL_STATIC_COPY`
`GL_DYNAMIC_DRAW`	`GL_DYNAMIC_READ`	`GL_DYNAMIC_COPY`
`GL_STREAM_DRAW`	`GL_STREAM_READ`	`GL_STREAM_COPY`

- Данные будут обновляться
  - `STATIC` - один раз
  - `DYNAMIC` - иногда
  - `STREAM` - почти каждый кадр
- Буфер будет использоваться для:
  - `DRAW` - записи данных в него
  - `READ` - чтения данных из него
  - `COPY` - и записи, и чтения

Буферы: параметр usage

`GL_STATIC_DRAW`	`GL_STATIC_READ`	`GL_STATIC_COPY`
`GL_DYNAMIC_DRAW`	`GL_DYNAMIC_READ`	`GL_DYNAMIC_COPY`
`GL_STREAM_DRAW`	`GL_STREAM_READ`	`GL_STREAM_COPY`

- Данные будут обновляться
  - `STATIC` - один раз
  - `DYNAMIC` - иногда
  - `STREAM` - почти каждый кадр
- Буфер будет использоваться для:
  - `DRAW` - записи данных в него
  - `READ` - чтения данных из него
  - `COPY` - и записи, и чтения
- Это только подсказка драйверу и не влияет на корректность работы

Буферы: запись и чтение данных

- Загрузить данные в часть буфера:

```
glBufferSubData(GLenum target, GLintptr offset,
    GLsizeiptr size, const GLvoid * data)
```

Буферы: запись и чтение данных

- Загрузить данные в часть буфера:

```
glBufferSubData(GLenum target, GLintptr offset,
    GLsizeiptr size, const GLvoid * data)
```

  - Гарантированно не реаллоцирует память GPU

Буферы: запись и чтение данных

- Загрузить данные в часть буфера:

```
glBufferSubData(GLenum target, GLintptr offset,
    GLsizeiptr size, const GLvoid * data)
```

  - Гарантированно не реаллоцирует память GPU
- Прочитать данные из буфера:

```
glGetBufferSubData(GLenum target, GLintptr offset,
    GLsizeiptr size, GLvoid * data)
```

Буферы: запись и чтение данных

- Загрузить данные в часть буфера:

```
glBufferSubData(GLenum target, GLintptr offset,
    GLsizeiptr size, const GLvoid * data)
```

  - Гарантированно не реаллоцирует память GPU
- Прочитать данные из буфера:

```
glGetBufferSubData(GLenum target, GLintptr offset,
    GLsizeiptr size, GLvoid * data)
```

  - К моменту выхода из этой функции данные уже прочитаны
  - $\Rightarrow$ Синхронная функция, блокирующая исполнение

Mapped buffer

- Можно получить виртуальный указатель на буфер или его часть и пользоваться им для чтения/записи

Mapped buffer

- Можно получить виртуальный указатель на буфер или его часть и пользоваться им для чтения/записи
- `glMapBuffer(target, access)` - возвращает mapped указатель
- `access` может принимать значения
  - `GL_READ_ONLY` - по указателю можно читать
  - `GL_WRITE_ONLY` - по указателю можно писать
  - `GL_READ_WRITE` - по указателю можно читать и писать

Mapped buffer

- Можно получить виртуальный указатель на буфер или его часть и пользоваться им для чтения/записи
- `glMapBuffer(target, access)` - возвращает mapped указатель
- `access` может принимать значения
  - `GL_READ_ONLY` - по указателю можно читать
  - `GL_WRITE_ONLY` - по указателю можно писать
  - `GL_READ_WRITE` - по указателю можно читать и писать
- `glUnmapBuffer(target)`

Mapped buffer

- Можно получить виртуальный указатель на буфер или его часть и пользоваться им для чтения/записи
- `glMapBuffer(target, access)` - возвращает mapped указатель
- `access` может принимать значения
  - `GL_READ_ONLY` - по указателю можно читать
  - `GL_WRITE_ONLY` - по указателю можно писать
  - `GL_READ_WRITE` - по указателю можно читать и писать
- `glUnmapBuffer(target)`
- Между `glMapBuffer` и `glUnmapBuffer` работать с буфером (загружать данные другими методами, использовать данные для рисования) нельзя

Mapped buffer

- Можно получить виртуальный указатель на буфер или его часть и пользоваться им для чтения/записи
- `glMapBuffer(target, access)` - возвращает mapped указатель
- `access` может принимать значения
  - `GL_READ_ONLY` - по указателю можно читать
  - `GL_WRITE_ONLY` - по указателю можно писать
  - `GL_READ_WRITE` - по указателю можно читать и писать
- `glUnmapBuffer(target)`
- Между `glMapBuffer` и `glUnmapBuffer` работать с буфером (загружать данные другими методами, использовать данные для рисования) нельзя
- После `glUnmapBuffer` mapped указатель использовать нельзя

Буферы: типичный пример использования

```
GLuint vbo;
glGenBuffers(1, &vbo);

std::vector<vertex> vertices;
...

glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER,
    vertices.size() * sizeof(vertices[0]),
    vertices.data(), GL_STATIC_DRAW);
```

Буферы: ссылки

- khronos.org/opengl/wiki/Buffer_Object
- songho.ca/opengl/gl_vbo.html