

Компьютерная графика

Лекция 15: оптимизация рендеринга, timer queries, frustum culling, occlusion culling, instancing

2021

Оптимизация – это сложно

На производительность (CPU) влияют:

Оптимизация – это сложно

На производительность (CPU) влияют:

- ▶ Общая загруженность системы

Оптимизация – это сложно

На производительность (CPU) влияют:

- ▶ Общая загрузка системы
- ▶ Количество и паттерн доступов к памяти (cache-friendliness)

Оптимизация – это сложно

На производительность (CPU) влияют:

- ▶ Общая загруженность системы
- ▶ Количество и паттерн доступов к памяти (cache-friendliness)
- ▶ Помещаются ли данные в кеш

Оптимизация – это сложно

На производительность (CPU) влияют:

- ▶ Общая загруженность системы
- ▶ Количество и паттерн доступов к памяти (cache-friendliness)
- ▶ Помещаются ли данные в кеш
- ▶ Branch prediction

Оптимизация – это сложно

На производительность (CPU) влияют:

- ▶ Общая загруженность системы
- ▶ Количество и паттерн доступов к памяти (cache-friendliness)
- ▶ Помещаются ли данные в кеш
- ▶ Branch prediction
- ▶ Как функции программы лежат в памяти (опять кеш)

Оптимизация – это сложно

На производительность (CPU) влияют:

- ▶ Общая загруженность системы
- ▶ Количество и паттерн доступов к памяти (cache-friendliness)
- ▶ Помещаются ли данные в кеш
- ▶ Branch prediction
- ▶ Как функции программы лежат в памяти (опять кеш)
- ▶ Многое другое

Оптимизация на GPU – это очень сложно

- ▶ Асинхронность

Оптимизация на GPU – это очень сложно

- ▶ Асинхронность
- ▶ Параллельность

Оптимизация на GPU – это очень сложно

- ▶ Асинхронность
- ▶ Параллельность
- ▶ Много встроенных операций (fixed-function pipeline)

Оптимизация на GPU – это очень сложно

- ▶ Асинхронность
- ▶ Параллельность
- ▶ Много встроенных операций (fixed-function pipeline)
- ▶ Сложные операции с памятью (доступ к текстуре: mipmaps + фильтрация)

Оптимизация на GPU – это очень сложно

- ▶ Асинхронность
- ▶ Параллельность
- ▶ Много встроенных операций (fixed-function pipeline)
- ▶ Сложные операции с памятью (доступ к текстуре: mipmaps + фильтрация)
- ▶ Многое другое

Измерение времени работы – неправильный способ

```
while (true) {  
    auto frame_start = clock::now();  
  
    // нарисовали сцену  
    ...  
  
    auto frame_end = clock::now();  
  
    SwapBuffers();  
}
```

Измерение времени работы – неправильный способ

```
while (true) {  
    auto frame_start = clock::now();  
  
    // нарисовали сцену  
    ...  
  
    auto frame_end = clock::now();  
  
    SwapBuffers();  
}
```

- ▶ `frame_end - frame_start` – сколько времени ушло на то, чтобы **вызвать OpenGL-команды**

Измерение времени работы – неправильный способ

```
while (true) {  
    auto frame_start = clock::now();  
  
    // нарисовали сцену  
    ...  
  
    auto frame_end = clock::now();  
  
    SwapBuffers();  
}
```

- ▶ `frame_end - frame_start` – сколько времени ушло на то, чтобы **вызвать OpenGL-команды**
- ▶ В реальности драйвер поставил их в очередь, и скорее всего GPU ещё не начала их выполнять

Измерение времени работы – простой способ

```
disableVsync();  
auto last_frame_start = clock::now();  
while (true) {  
    auto frame_start = clock::now();  
    auto frame_time = frame_start - last_frame_start;  
    last_frame_start = frame_start;  
  
    // нарисовали сцену  
    ...  
  
    SwapBuffers();  
}
```

Измерение времени работы – простой способ

```
disableVsync();  
auto last_frame_start = clock::now();  
while (true) {  
    auto frame_start = clock::now();  
    auto frame_time = frame_start - last_frame_start;  
    last_frame_start = frame_start;  
  
    // нарисовали сцену  
    ...  
  
    SwapBuffers();  
}
```

- ▶ Из-за выключенного vsync видеокарта будет работать ± постоянно

Измерение времени работы – простой способ

```
disableVsync();  
auto last_frame_start = clock::now();  
while (true) {  
    auto frame_start = clock::now();  
    auto frame_time = frame_start - last_frame_start;  
    last_frame_start = frame_start;  
  
    // нарисовали сцену  
    ...  
  
    SwapBuffers();  
}
```

- ▶ Из-за выключенного vsync видеокарта будет работать \pm постоянно
- ▶ В итоге мы получим примерное время, тратящееся на рисование одного кадра

Измерение времени работы: glFlush и glFinish

- ▶ Многие (старые) tutorиалы по измерению времени кадра советуют вызывать `glFlush` или `glFinish` в конце кадра

Измерение времени работы: glFlush и glFinish

- ▶ Многие (старые) tutorиалы по измерению времени кадра советуют вызывать glFlush или glFinish в конце кадра
- ▶ glFlush сбрасывает буфер команд (хранящийся внутри драйвера) с CPU на GPU

Измерение времени работы: glFlush и glFinish

- ▶ Многие (старые) tutorиалы по измерению времени кадра советуют вызывать glFlush или glFinish в конце кадра
- ▶ glFlush сбрасывает буфер команд (хранящийся внутри драйвера) с CPU на GPU
- ▶ glFinish ждёт, пока GPU не завершит обрабатывать все посланные команды

Измерение времени работы: glFlush и glFinish

- ▶ Многие (старые) tutorиалы по измерению времени кадра советуют вызывать glFlush или glFinish в конце кадра
- ▶ glFlush сбрасывает буфер команд (хранящийся внутри драйвера) с CPU на GPU
- ▶ glFinish ждёт, пока GPU не завершит обрабатывать все посланные команды
- ▶ SwapBuffers сама вызывает glFlush

Измерение времени работы: glFlush и glFinish

- ▶ Многие (старые) tutorиалы по измерению времени кадра советуют вызывать glFlush или glFinish в конце кадра
- ▶ glFlush сбрасывает буфер команд (хранящийся внутри драйвера) с CPU на GPU
- ▶ glFinish ждёт, пока GPU не завершит обрабатывать все посланные команды
- ▶ SwapBuffers сама вызывает glFlush
- ▶ glFinish ухудшает производительность: половину времени вы отправляете команды на GPU, а GPU (скорее всего) ничего не делает; половину времени вы ждёте, пока GPU закончит выполнять команды

Измерение времени работы: FPS vs frame duration

- ▶ FPS (frames per second, количество кадров в секунду) – очень неудобная метрика:

Измерение времени работы: FPS vs frame duration

- ▶ FPS (frames per second, количество кадров в секунду) – очень неудобная метрика:
 - ▶ Нелинейна: если кадр рисовался 10 мс, и мы добавили что-то рисующееся 1 мс, и ещё что-то рисующееся 1 мс, то FPS изменялся от 100 до 90.9 до 83.3

Измерение времени работы: FPS vs frame duration

- ▶ FPS (frames per second, количество кадров в секунду) – очень неудобная метрика:
 - ▶ Нелинейна: если кадр рисовался 10 мс, и мы добавили что-то рисующееся 1 мс, и ещё что-то рисующееся 1 мс, то FPS изменялся от 100 до 90.9 до 83.3
- ▶ Обычно используют время, тратящееся на рисование кадра или конкретного объекта/эффекта (миллисекунды/микросекунды)

Измерение времени работы – правильный способ: timer queries

- ▶ Query objects – объекты OpenGL, позволяющие узнать некоторую статистику с GPU:

Измерение времени работы – правильный способ: timer queries

- ▶ Query objects – объекты OpenGL, позволяющие узнать некоторую статистику с GPU:
 - ▶ Сколько было нарисовано пикселей

Измерение времени работы – правильный способ: timer queries

- ▶ Query objects – объекты OpenGL, позволяющие узнать некоторую статистику с GPU:
 - ▶ Сколько было нарисовано пикселей
 - ▶ Сколько сгенерировано примитивов (геометрическим шейдером)

Измерение времени работы – правильный способ: timer queries

- ▶ Query objects – объекты OpenGL, позволяющие узнать некоторую статистику с GPU:
 - ▶ Сколько было нарисовано пикселей
 - ▶ Сколько сгенерировано примитивов (геометрическим шейдером)
 - ▶ Сколько прошло времени

Измерение времени работы – правильный способ: timer queries

- ▶ Query objects – объекты OpenGL, позволяющие узнать некоторую статистику с GPU:
 - ▶ Сколько было нарисовано пикселей
 - ▶ Сколько сгенерировано примитивов (геометрическим шейдером)
 - ▶ Сколько прошло времени
- ▶ `glGenQueries/glDeleteQueries`

Измерение времени работы – правильный способ: timer queries

- ▶ Query objects – объекты OpenGL, позволяющие узнать некоторую статистику с GPU:
 - ▶ Сколько было нарисовано пикселей
 - ▶ Сколько сгенерировано примитивов (геометрическим шейдером)
 - ▶ Сколько прошло времени
- ▶ `glGenQueries/glDeleteQueries`
- ▶ **Нет** `glBindQuery!`

Измерение времени работы – правильный способ: timer queries

- ▶ `glBeginQuery/glEndQuery` – статистика будет собрана для команд между этими вызовами

Измерение времени работы – правильный способ: timer queries

- ▶ `glBeginQuery/glEndQuery` – статистика будет собрана для команд между этими вызовами
- ▶ **Не могут** быть вложенными

Измерение времени работы – правильный способ: timer queries

- ▶ glBeginQuery/glEndQuery – статистика будет собрана для команд между этими вызовами
- ▶ **Не могут** быть вложенными

```
GLuint query_id;  
glGenQueries(1, &query_id);  
  
...  
  
glBegin(GL_TIME_ELAPSED, query_id);  
  
// что-нибудь рисуем  
  
glEnd(GL_TIME_ELAPSED);
```

Измерение времени работы – правильный способ: timer queries

- ▶ GPU работает асинхронно \Rightarrow результат query будет готов не сразу

Измерение времени работы – правильный способ: timer queries

- ▶ GPU работает асинхронно \Rightarrow результат query будет готов не сразу
- ▶ Узнать, готов ли результат:

```
glGetQueryObjectiv(query_id,  
    GL_QUERY_RESULT_AVAILABLE, &result);
```

Измерение времени работы – правильный способ: timer queries

- ▶ GPU работает асинхронно \Rightarrow результат query будет готов не сразу

- ▶ Узнать, готов ли результат:

```
glGetQueryObjectiv(query_id,  
GL_QUERY_RESULT_AVAILABLE, &result);
```

- ▶ Получить результат (блокирует поток, если результат ещё не готов; неявно вызывает glFlush)

```
glGetQueryObjectiv(query_id,  
GL_QUERY_RESULT, &result);
```

Измерение времени работы – правильный способ: timer queries

- ▶ GPU работает асинхронно \Rightarrow результат query будет готов не сразу

- ▶ Узнать, готов ли результат:

```
glGetQueryObjectiv(query_id,  
GL_QUERY_RESULT_AVAILABLE, &result);
```

- ▶ Получить результат (блокирует поток, если результат ещё не готов; неявно вызывает glFlush)

```
glGetQueryObjectiv(query_id,  
GL_QUERY_RESULT, &result);
```

- ▶ Время возвращается в **наносекундах**, т.е. знаковый 32-битный тип может представить 2 секунды

Измерение времени работы – правильный способ: timer queries

- ▶ GPU работает асинхронно \Rightarrow результат query будет готов не сразу

- ▶ Узнать, готов ли результат:

```
glGetQueryObjectiv(query_id,  
GL_QUERY_RESULT_AVAILABLE, &result);
```

- ▶ Получить результат (блокирует поток, если результат ещё не готов; неявно вызывает glFlush)

```
glGetQueryObjectiv(query_id,  
GL_QUERY_RESULT, &result);
```

- ▶ Время возвращается в **наносекундах**, т.е. знаковый 32-битный тип может представить 2 секунды
- ▶ Если 64-битные и беззнаковые версии этих функций

Измерение времени работы – правильный способ: пул timer queries

- ▶ Хотим мерять время рисования каждого кадра, но результат для предыдущего кадра может быть не готов к началу следующего кадра

Измерение времени работы – правильный способ: пул timer queries

- ▶ Хотим мерять время рисования каждого кадра, но результат для предыдущего кадра может быть не готов к началу следующего кадра
- ▶ \Rightarrow Заводим пул (pool) query-объектов:

Измерение времени работы – правильный способ: пул timer queries

- ▶ Хотим мерять время рисования каждого кадра, но результат для предыдущего кадра может быть не готов к началу следующего кадра
- ▶ \Rightarrow Заводим пул (pool) query-объектов:
 - ▶ Храним расширяемый массив (`std::vector`) query-объектов: ID + свободен или нет

Измерение времени работы – правильный способ: пул timer queries

- ▶ Хотим мерять время рисования каждого кадра, но результат для предыдущего кадра может быть не готов к началу следующего кадра
- ▶ \Rightarrow Заводим пул (pool) query-объектов:
 - ▶ Храним расширяемый массив (`std::vector`) query-объектов: ID + свободен или нет
 - ▶ Когда нам нужен новый query, ищем в массиве свободный объект, если такого нет - добавляем новый

Измерение времени работы – правильный способ: пул timer queries

- ▶ Хотим мерять время рисования каждого кадра, но результат для предыдущего кадра может быть не готов к началу следующего кадра
- ▶ \Rightarrow Заводим пул (pool) query-объектов:
 - ▶ Храним расширяемый массив (`std::vector`) query-объектов: ID + свободен или нет
 - ▶ Когда нам нужен новый query, ищем в массиве свободный объект, если такого нет - добавляем новый
 - ▶ В конце рисования кадра проходим по всем несвободным объектам и проверяем: если результат уже готов, обрабатываем его и помечаем объект свободным

Измерение времени работы – правильный способ: пул timer queries

- ▶ Хотим мерять время рисования каждого кадра, но результат для предыдущего кадра может быть не готов к началу следующего кадра
- ▶ \Rightarrow Заводим пул (pool) query-объектов:
 - ▶ Храним расширяемый массив (`std::vector`) query-объектов: ID + свободен или нет
 - ▶ Когда нам нужен новый query, ищем в массиве свободный объект, если такого нет - добавляем новый
 - ▶ В конце рисования кадра проходим по всем несвободным объектам и проверяем: если результат уже готов, обрабатываем его и помечаем объект свободным
- ▶ Средний размер пула – на сколько кадров отстаёт GPU от CPU

Timer queries: ссылки

- ▶ [khronos.org/opengl/wiki/Query_Object](https://www.khronos.org/opengl/wiki/Query_Object)
- ▶ Тьюториал по использованию timer queries