Авторы: Щербаков А.С, Фролов В.А.

Задание 3. Симуляция и рендеринг растительности.

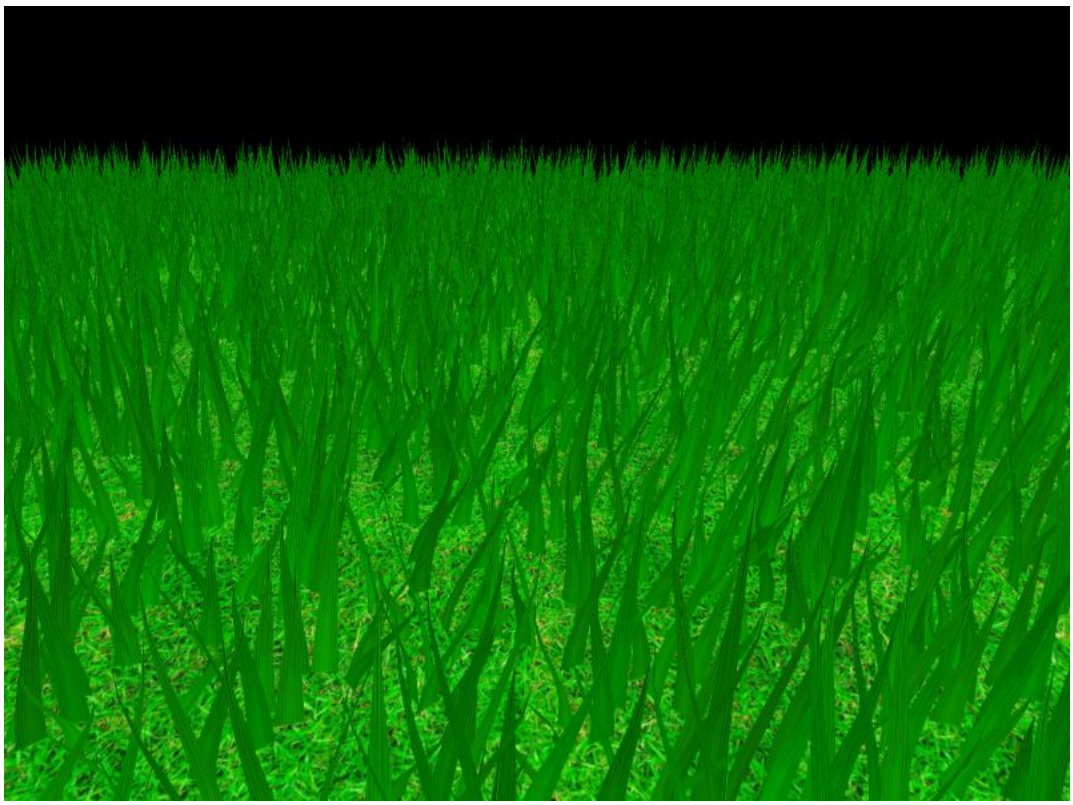

Рисунок 1. Пример выполненного задания.

1.Аннотация

Цель задания: Изучение основ современного OpenGL в процессе выполнения задания с полезной нагрузкой:

База:

- Основы физической симуляции и анимации
- Графический конвейер, геометрические преобразования
- Шейдерные программы
- VBO/VAO. Объекты памяти OpenGL
- Инстансинг

Дополнительно:

- Локальные модели освещения
- Модели освещения тонких объектов
- Имитация микрорельефа
- Тени. Карты теней.
- Редеринг в текстуру
- Научная визуализация

Предполагаемый объем кода задания: 1000-1500 строк:

- Инициализация меша травы 100 строк.
- Симуляция: 100 строк.
- Работа с OpenGL: 500-1000 строк.

Пример выполненного задания: https://youtu.be/QNTAT74Pddk

2.База

В базовой реализации задания необходимо сделать траву (или другую растительность - цветы, кусты), реалистично колышущуюся на ветру (рис. 1). По желанию допускается реализация волос вместо травы либо деревьев/кустов с колышущимися на ветру листьями. Отрисовка травы должна выполняться при помощи механизма инстансинга в OpenGL.

2.1 Обязательные требования к симуляции:

- Движения травы должны быть реалистичными. Вся трава целиком должна шевелиться (а не только отдельные травинки).
- Ветер должен быть включён по умолчанию, либо добавляться по кнопке ‘W’ (Wind).
- Скорость симуляции **не должна зависеть** от производительности машины. Можно включить вертикальную синхронизацию, чтобы гарантировать ограничение 60 кадров в секунду. На машине проверяющего вертикальная синхронизация будет включена по умолчанию.

  Совет: запускайте свое задание на разных по производительности GPU!

2.2 Обязательные требования рендерингу:

- Травинки (либо другие элементы растительности) должны быть похожи на настоящую траву - сделаны геометрией либо плашками с текстурами как в [1] или [3]. Простые прямоугольные полоски, один треугольник, линии или точки, схематично изображающие траву не будут засчитаны!

- Отдельные травинки (либо другие элементы растительности) должны иметь различный размер и поворот (возможно также разный цвет).

- Минимальное количество инстанцируемых объектов (травинок) - 10000. В случае если вместо травинок используются более сложные объекты (например целые деревья или кусты), минимальное количество инстанцируемых объектов - 5000.

- Отрисовка травы должна выполняться при помощи механизма инстансинга [13].

- В сцене должна быть текстурированная плоскость, имитирующая пол. Аналогично тому как сделано в примере. Либо какая-то более сложная имитация земли.

- В случае реализации волос обязательна визуализация головы хотя бы сферой и взаимодействие волос с ней (т.е. волосы не должны проходить сквозь голову). В случае реализации сценария с деревьями - обязательна визуализация текстурированного ствола дерева.

3. Критерии оценки и дополнительная часть

Максимальная оценка за задание – 40 баллов.

База. Симуляция и визуализация травы и земли с текстурой в соответствии с требованиями, без освещения (10 баллов).

Дополнительно. Дополнительная часть задания (максимум 30 баллов).

- Анти-алиасинг (До +5)
  - Должен включаться и отключаться по кнопке “A”.
  - MSAA (+2) (http://www.learnopengl.com/#!Advanced-OpenGL/Anti-aliasing)
  - Super Sampling (+3) - рендеринг в текстуру с бОльшим разрешением (чем разрешение экрана) и даунсэмплинг
  - Super Sampling + MSAA (+5)

- Базовое освещение (до + 5 баллов).
  - Текстура на травинках (+1).
  - Освещение по ламберту и/или фонгу с одним источником света (+1).
  - Два разных источника света (например направленный и точечный) (+1).
  - Затемнение травы ближе к земле для имитации эффекта Ambient Occlusion (+1).
  - Имитация грязи и/или пожелтения (+1)
  - Специализированные модели освещения травы или волос (+2)

- Тени (до + 6 баллов).
  - Тени на плоскость (+1)
  - Простой Shadow Map (+3)
  - Простой Shadow Map + PCF (+4)
  - Более сложные и реалистичные методы теней (до +6)

  По кнопке '7' должна включатся визуализация карты глубины, которая строится при рендере сцены из источника. Визуализация карты глубины должна позволять различать в ней отдельные объекты (карта может быть не однотонной, либо можно не рисовать плоскость в карту глубины).

  При реализации карт теней необходимо добавить хотя бы 1 дополнительный объект в сцену, чтобы тень падала от травы на этот объект, либо от объекта на траву, либо должно быть реализовано и хорошо видно самозатенение травы.

- Более реалистичная симуляция на CPU (до +4).

    - Отдельная симуляция нескольких (10-100) травинок и цветов по разному (до +2 в зависимости от реалистичности).
    - Более сложные модели симуляции (до +4 в зависимости от реалистичности) [2].

- Более реалистичная симуляция на GPU (до +5 баллов).

  Учёт столкновения с простыми объектами, перемещающимися по траве (например, катающийся по траве мяч). Данный пункт необходимо реализовывать полностью на GPU (при помощи OpenGL, OpenCL или CUDA), поскольку объекты будет сталкиваться только с некоторыми травинками и все травинки должны реагировать на это столкновение по разному.

- Добавление дополнительных объектов (До +8 баллов):
    - 2-3 различных типа растительности (например, трава и 2 разных вида цветочков) (+1 за каждый дополнительный вид, +2 максимум)
    - Добавление деревьев (+2 за статичные деревья +4 за деревья с колышущимися на ветру листьями)
    - Добавление животных или птиц (+1 за каждый вид, +2 максимум)
    - Добавление камней (+1 максимум)
    - Добавление любых других дополнительных объектов (+1 максимум)

- Окружение (До +8 баллов)
    - Ландшафт вместо плоскости. Геометрию ланшафта можно сгенерировать по карте высот. Трава должна расти при этом на поверхности ландшафта (+2)
    - Кубические текстурные карты для визуализации неба (+1)
    - Добавление атмосферных эффектов при помощи частиц (дождь, снег, пепел … - до +2 в зависимости от реалистичности).
    - Добавление воды
        - (+1 просто плоского участка воды)
        - (+2 вода с отражениями)
        - (+3 вода с отражениями и прозрачностью)

- Имитация микрорельефа на земле, камнях, травинке или коре деревьев (normal mapping) (до +6 в зависимости от техники и реалистичности)
    - +2 простой нормал-маппинг (бамп-маппинг)
    - +4 Parallax Occlusion Mapping
    - +6 Использование тесселяции (по кнопке ‘g’ должна включаться каркасная визуализация протесселированнйо поверхности !!! + должна быть возможность включать/отключать тесселяцию по кнопке ‘t’)

- Научная визуализация (до +5)
    - Визуализация сил, действующих на вершины сетки в виде стрелочек (до +2 в зависимости от наглядности и эстетичности).
    - Визуализация числовых значений величин (сил, скоростей, позиций) по кнопкам ('8', '9', '0') в точке куда указывает мышка. (+3)

Подсказка 1: Самый простой способ - использовать рендеринг в отдельные текстуры сил, скоростей и позиций. См. пример 'basic/sample_10_mrt' [0].
Затем можно выводить числовые значения в виде 3 чисел в названия окна.

4. Материалы для выполнения задания

Для облегчения выполнения задания мы описываем некоторые существующие методы и даём ссылки на литературу. Это тем не менее не означает что вы обязаны использовать именно эти методы. Вы можете использовать любые техники алгоритмы, которые позволят Вам получить реалистичный результат.

4.0 Модель элемента растительности

Следует отметить, что физическая модель не обязана совпадать с геометрической полигональной моделью, используемой для рендеринга. Чаще всего они не совпадают. В данном задании мы предлагаем реализовать Вам следующие простые модели:

4.0.1 Физическая модель

Для задания была выбрана достаточно простая физическая модель. Ветер сообщает каждой травинке некоторую скорость. Чем выше точка модели, тем больше полученная скорость. Под действием этой скорости точки меняют своё положение в направлении вектора скорости. Так как травинка деформируется, то появляется сила упругости, которая пропорциональна изменению положения точки. Вектор силы направлен от текущего положения точки к первоначальному.

4.0.2 Геометрическая модель

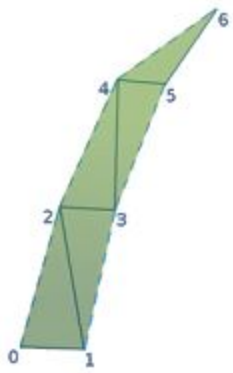

Рисунок 2. Простая геометрия травинки.

В данном задании вы можете использовать любую реалистично-выглядящую модель травы. Мы предлагаем Вам несколько вариантов геометрических моделей:

- Простая полигональная модель (рис.2)
- Полигональная модель с текстурой для альфа-теста (рисунки A и B)
- Полигональная модель на основе плашек (рисунок C).

Один из способов уменьшения количества полигонов (при сохранении детальной модели) - использование текстуры альфы и выбрасывание ненужных участков травы прямо во фрагментном шейдере при помощи инструкции discard (рисунок A).

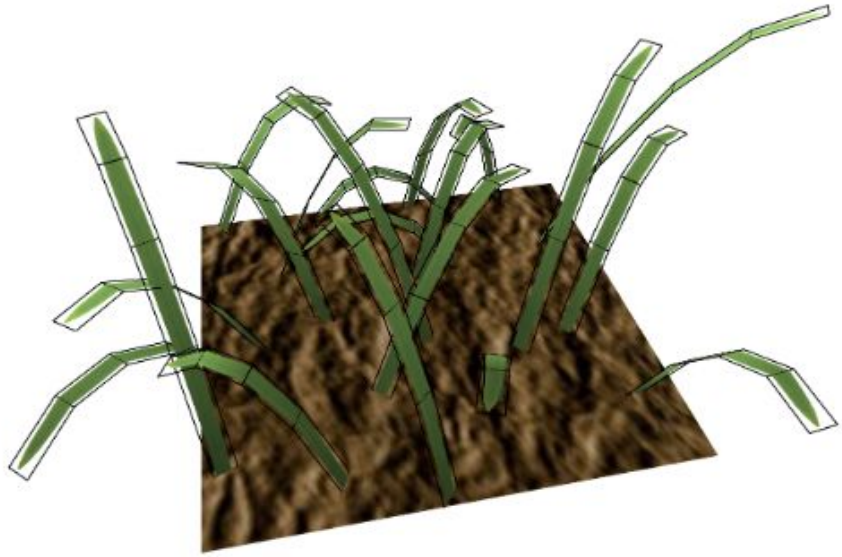

Рисунок 3. Простая модель растительности на основе геометрии [11].

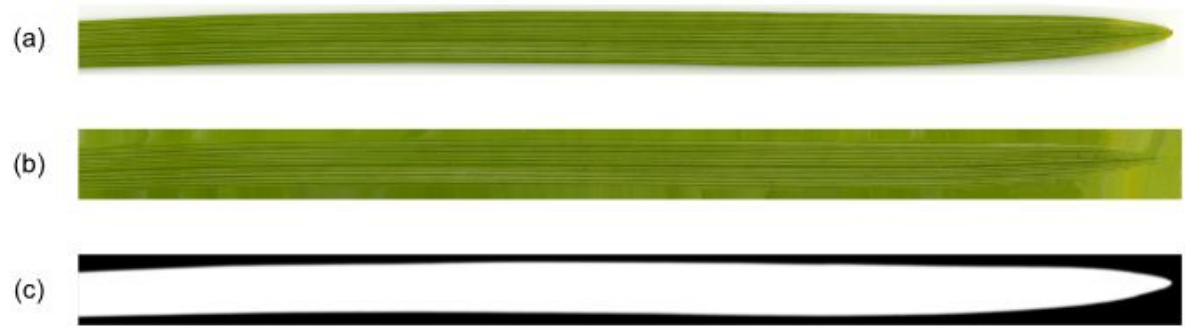

Рисунок 4. Используемые текстуры (b - цвет; c - альфа; a - результат применения альфа-теста при помощи инструкции **discard** во фрагментном шейдере) [11].

Внимание!!! - при отрисовке травы **не используйте прозрачность** !!!

Для корректной визуализации прозрачной травы при помощи альфа-смешивания Вам придется сортировать все полигоны травы от дальнего к ближнему непосредственно на GPU, что само по себе является довольно сложной задачей. Но если всё-таки Вам очень хочется это сделать, для сортировки вы можете использовать простой алгоритм bitonic sort на OpenCL [12]. Альтернативное решение - использование так называемой порядко-независимой прозрачности (Order Independent Transparency) - попиксельная сортировка отдельных фрагментов в вычислительном шейдере [13].

Также можно использовать модель на основе плашек с текстурами [1,3] (рисунок 5).

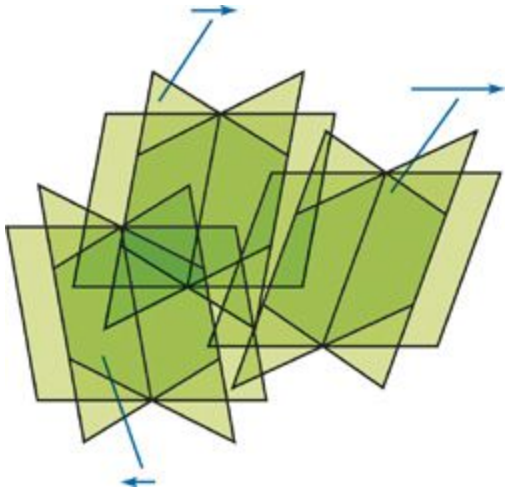

Рисунок 5. Альтернативная модель растительности на основе плашек [1,3].

4.1 Основы симуляции движений

В контексте данного задания система - набор вершин физической модели травинок. На каждом кадре вы должны вычислять физические параметры в вершинах - позицию, скорость, силы. Исходя из параметров на текущем кадре вы вычисляете параметры на следующем кадре.

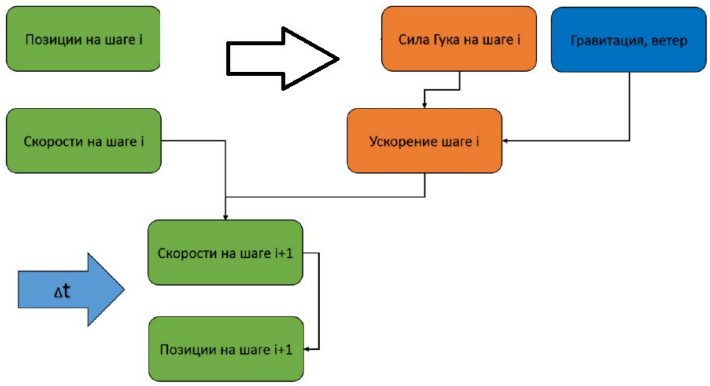

Рисунок 6. Пошаговая симуляция движений. Сила Гука - сила упругости, действующая на точки вашей модели (если в модели имеются пружины или что-то подобное).

4.2 Основы геометрических преобразований в современном OpenGL

1. При использовании OpenGL3/4 и шейдеров необходимо помнить одно главное правило. Вершинный шейдер записывает свой выход в gl_Position в clip space, в единичном кубе [-1,1].
2. Также важно помнить, что в процессе растеризации происходит деление позиций вершин на четвёртую координату w. Это позволяет записать перспективную проекцию в виде матрицы (без перехода в 4 измерения это было бы невозможно). Поэтому четвёртая координата (w) должна быть равна 1 до применения матрицы проекции. И только после применения матрицы проекции она может быть не равной 1.

3. Помните, что драйвер OpenGL обычно выполняет оптимизации неиспользуемых атрибутов вершин. Поэтому если в вершинном шейдере какой-нибудь атрибут не используется, его location получается инвалидным.

4.3 Инстансинг в современном OpenGL

Инстансинг - это возможность рисовать множество однотипных объектов при помощи одного вызова OpenGL функции. Нарисовать множество объектов в действительности можно и без механизма инстансинга, например в цикле:

```
glBindVertexArray(bladeVAO);
for(int i = 0; i < 100; i++)
{
    // bind Textures, set uniforms etc.
    // …
    glDrawArrays(GL_TRIANGLES, 0, 10);
}
```

Однако это не эффективно, если объекты относительно низкополигональные и их много. Трава - как раз такой случай. Вместо этого можно рисовать всю траву всего двумя вызовами:

```
glBindVertexArray(grassVAO);
glDrawArraysInstanced(GL_TRIANGLES, 0, 10, 100);
```

Обратите внимание, что bladeVAO и grassVAO - разные, хотя и похожие объекты типа VertexArrayObject. bladeVAO должен содержать только указатели на данные для одной травинки, в то время как grassVAO - такой же как bladeVAO, но дополнительно содержит указатель на буфер для данных на один инстанс (например позиций травинок) - рис.7.

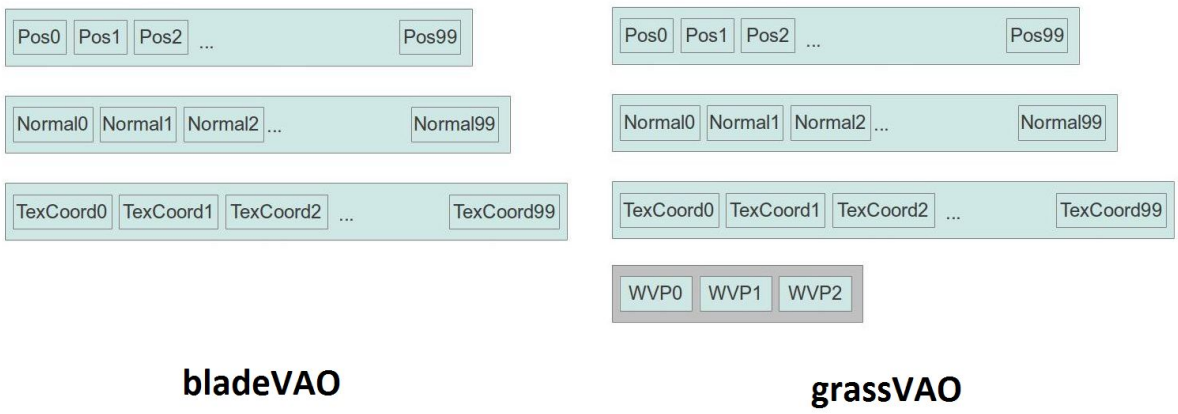

Рисунок 7. Иллюстрация VAO для одной травинки (bladeVAO) и VAO для массива травы целиком (grassVAO). bladeVAO содержит 3 указателя на VBO хранящих вершины, в то время как grassVAO содержит те же 3 указателя + указатель на буфер матриц. Для того чтобы определить, как именно будет происходить передача атрибутов (“один элемент массива на 1

вершину” либо “один элемент массива на целый объект”) используется функция glvertexAttribDivisor (рис.7):

```
// Attrib 4 (vertex information) changes per vertex
glVertexAttribDivisor(4,0)

// Attrib 5 (matrix data) changes per instance
glVertexAttribDivisor(5,1)
```

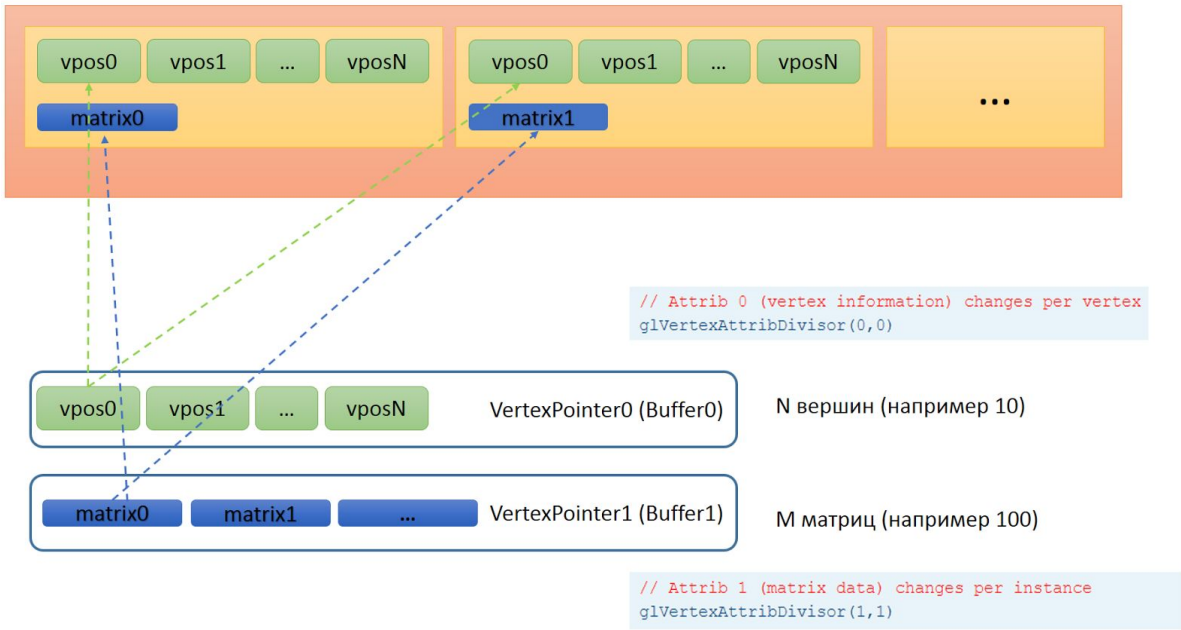

Рисунок 8. Иллюстрация передачи данных в вершинный шейдер при работе инстансинга. При использовании glVertexAttribDivisor(*,0) данные vpos0..vposN будут переданы в вершинный шейдер по правилу “один элемент массива на 1 вершину” (per-vertex). При использовании glVertexAttribDivisor(*,1) данные будут передаваться в вершинный шейдер по правилу “один элемент массива на один инстанс” (per-instance).

Подсказка: Если вы хотите передавать матрицы per-instance как атрибуты вершины, вам необходимо разбить ваш буфер матриц по строкам на 4 разных буфера (т.е. буфер из элементов по 16 float представить как 4 буфера из элементов по 4 float). Данное ограничение связано с так называемым объединением запросов в память (coalescing), когда GPU может эффективно читать определённые последовательности данных (например float2 или float4). Оно введено в OpenGL как механизм защиты от потенциального формата данных, который может работать неэффективно (что может на самом деле зависеть от модели GPU).

Более подробную информацию про инстансинг можно получить пройдя по ссылкам [14] и [17].

4.4 Основы локальных моделей освещения

В OpenGL 3/4 нет встроенных моделей освещения. Все расчёты освещенности фрагмента/пиксела вы выполняете самостоятельно при помощи вершинных и фрагментных

программ (как правило освещение считается при помощи последних). Для расчёта освещённости во фрагментном шейдере предлагается использовать модели Ламберта и Фонга.

Важно помнить, что все расчёты вы должны производить в одном и том же пространстве. Например, если вы всё считаете в мировом пространстве (world space), не забудьте правильно вычислить направление на наблюдателя и источник света.

Если вы выполняете расчёт освещения в пространстве камеры (в этом пространстве легко получить направление на наблюдателя), не забудьте корректно перевести в это пространство направление на источник.

Если вы выполняете расчёт в тангенциальном пространстве (tangent space) для имитации микрорельефа при помощи карт нормалей, необходимо все вектора перевести в тангенциальное пространство.

4.5 Метод карт теней

Идея метода карт теней заключается в том, чтобы запомнить буфер глубины при рендере с позиции источника. В дальнейшем, зная расстояние от точки до источника, и зная это же расстояние в буфере глубины, можно определить, лежит точка в тени или нет (рисунки 9 и 10).

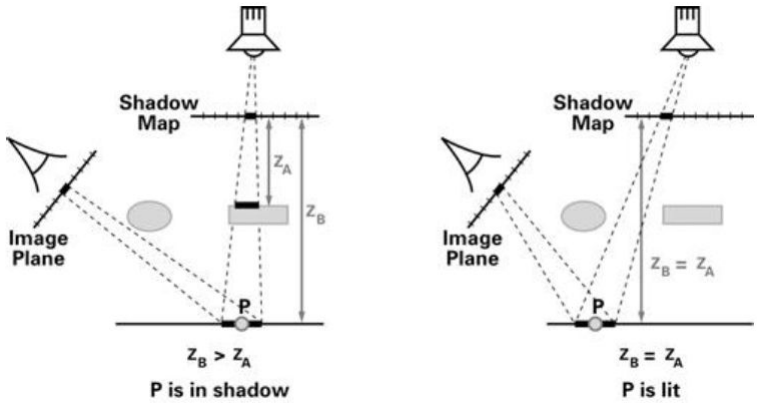

Рисунок 9. Иллюстрация работы метода карт теней.

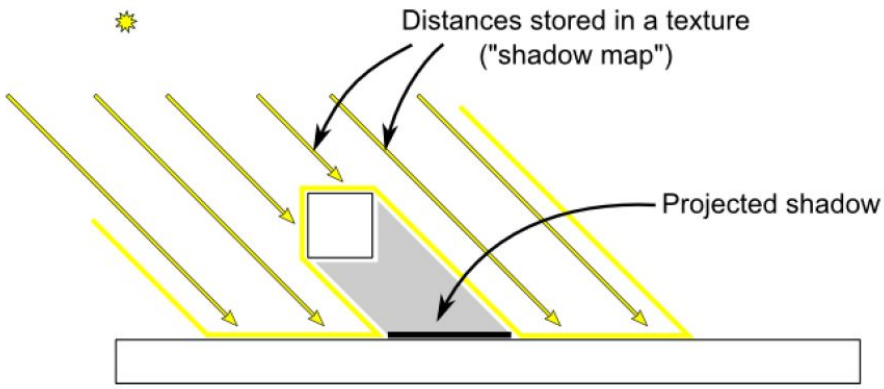

Рисунок 10. Ещё одна иллюстрация работы метода карт теней.

Если расстояние от заданной точки до источника в реальности больше, чем это же самое расстояние, запомненное в буфере глубины, значит, данная точка в тени, т.к. это расстояние

может быть меньше только если на пути света к этой точки из источника встретилось препятствие. Таким образом, метод карт теней состоит из 4 шагов:

1. Расчёт модельно-видовой матрицы для вида из источника освещения.
2. Расчёт матрицы проекции для вида из источника освещения.
3. Рендер сцены из позиции источника и сохранение глубины в отдельной текстуре.
4. При основном рендере с позиции камеры для каждого пиксела нужно перевести его позиции из мирового пространства (world space) в пространство источника света (view space) и получить значение глубины. Вы делаете это при помощи ровно тех же самых матриц (модельно-видовой матрицы и матрицы проекции), которые были использованы при рендере сцены из позиции источника освещения.

Помните также, что если вы используете перспективную проекцию при рендере сцены из позиции источника освещения, после приминения матрицы проекции на 4 шаге вы должны поделить получившиеся координаты на w (алгоритм 2).

```
vec4 posLightSpace  = shadowViewMatrix*vec4(fragmentWorldPos, 1);
vec4 posClipSpace   = shadowProjMatrix*posLightSpace;
vec2 shadowTexCoord = (posClipSpace.xy/posClipSpace.w)*0.5 + vec2(0.5, 0.5);
```

Алгоритм 2. Перевод позиции фрагмента из world space в light space и получение текстурных координат для выборки глубины из карты теней.

4.6 Рендеринг в текстуру в OpenGL

Рендеринг в текстуру в OpenGL реализуется при помощи абстракции, называемой Frame Buffer Object (FBO) (рис 9). Как только вы делаете фрейм-буфер текущим/активным, весь рендеринг с экрана перенаправляется в прикреплённые к фрейм-буферу текстуры. В предоставляемом Вам шаблоне будет небольшой класс RenderableTexture2D, который реализует работу с framebuffer-ом OpenGL.

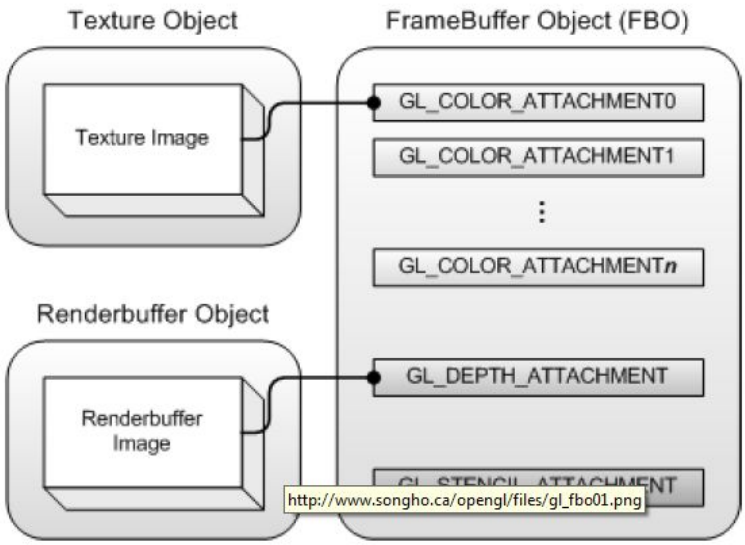

Рисунок 9. Иллюстрация работы Frame Buffer Object (FBO). Хотя на данном рисунке к GL_DEPTH_ATTACHMENT прикреплён специальный объект renderbuffer, вместо него может быть текстура специального формата. Именно так сделано в предоставляемом вам шаблоне.

5.Правила оформления работы

Внимание! При невыполнении указанных требований работа может не проверяться! Архив с заданием в формате zip должен быть залит в систему курса. В случае превышения максимального размера архива в системе нужно разбить его на части средствами архиватора. Заливать архив на файлообменники можно только в случае невозможности залить его в систему, по предварительному согласованию с проверяющими.

Содержимое архива:

1. Папка src (исходный код)
    a. Файлы исходного кода
    b. Файлы проекта
    c. Для тех кто пользуется Visual Studio - НЕ нужно включать в архив папку ipch, базы данных программы .ncb, .sdf.
    d. Проект должен собираться из папки src
    e. Для линукса необходимо наличие make файла если использовалась отличная от CodeBlocks среда сборки.
2. Папка bin (исполняемый код - конфигурация Release, 32 бит). Обязательно проверьте, что программа запускается из папки bin. Желательно, на другой машине.
    a. Исполняемый файл
    b. Библиотеки, необходимые для запуска
    c. Данные (модели, текстуры, файл настроек). Дублировать данные в папке src не нужно.
3. Файл Readme.txt
    a. Фамилия, имя, отчество, группа
    b. Операционная система

c. Инструкцию по сборке если сборка нетривиальна

6.Литература

[0] MSU Graphics & Media Lab OpenGL4 SDK. URL = https://github.com/FROL256/msu-opengl4-sdk

[1] Kurt Pelzer. Rendering Countless Blades of Waving Grass. GPU Gems. Chapter 7. Piranha Bytes.Addison-Wesley. 2007.URL = http://http.developer.nvidia.com/GPUGems/gpugems_ch07.html

[2] By Changbo Wang*, Zhangye Wang, Qi Zhou, Chengfang Song, Yu Guan and Qunsheng Peng. Dynamic modeling and rendering of grass wagging in wind.

[3] Боресков А. В. Рендеринг травы. http://steps3d.narod.ru/tutorials/grass-tutorial.html

[4] Уроки OpenGL3. https://code.google.com/p/gl33lessons/

[5] Боресков А.В. Статьи по 3D графике. URL = http://steps3d.narod.ru/articles.html

[6] OpenGL Programming Guide. The official guide to learning OpenGL*, version 4.3 8 издание. URL = http://www.ics.uci.edu/~gopi/CS211B/opengl_programming_guide_8th_edition.pdf

[7] Более неформальный цикл уроков по OpenGL4 http://triplepointfive.github.io/ogltutor/index.html

[8] Jason L. McKesson. Learning Modern 3D Graphics Programming http://www.pdfiles.com/pdf/files/English/Designing_&_Graphics/Learning_Modern_3D_Graphics_Programming.pdf

[10] Norbert Nopper’s samples https://github.com/McNopper

[11] https://hal.inria.fr/file/index/docid/87776/filename/grassRR.pdf

[12] OpenCL Bitonic Sort. https://github.com/FROL256/opencl_bitonic_sort_by_key

[13] Order Independent Transparency URL = http://on-demand.gputechconf.com/gtc/2014/presentations/S4385-order-independent-transparency-opengl.pdf

[14] Инстансинг в OpenGL. http://learnopengl.com/#!Advanced-OpenGL/Instancing

[15] Документация по glut. https://www.opengl.org/resources/libraries/glut/spec3/spec3.html

[16] Документация по freeglut. http://freeglut.sourceforge.net/docs/api.php

[17] Ещё инстансинг https://triplepointfive.github.io/ogltutor/tutorials/tutorial33.html