Авторы: *Владимир Фролов, Владимир Афанасьев.*

Задание 4
Симуляция и визуализация тканей

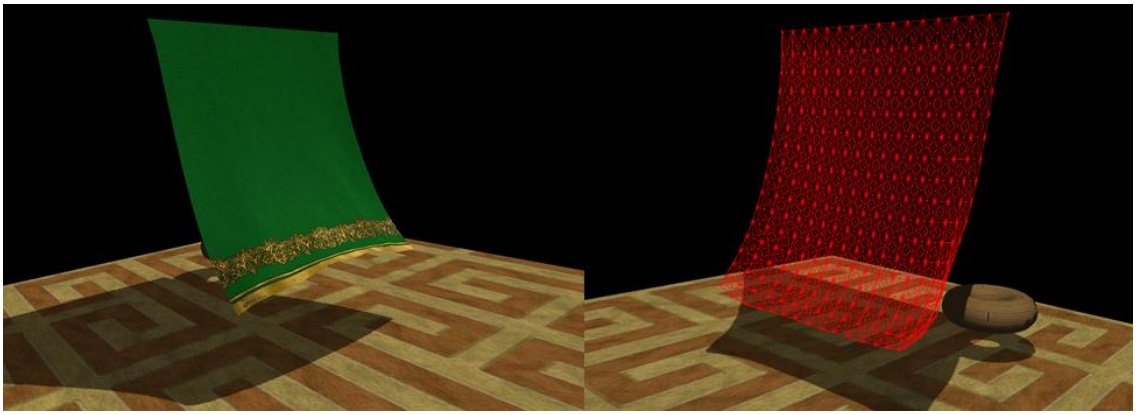

Рисунок 1. Пример выполненного задания.

1 Аннотация

Цель задания: Изучение основ OpenGL3/4 в процессе выполнения задания с полезной нагрузкой.

- Основы метода конечных элементов (МКЭ) (база).
- Графический конвейер, геометрические преобразования (база).
- Шейдерные программы (база).
- VBO/VAO. Объекты памяти OpenGL (база).
- Отладочная визуализация. Точки и линии (база).
- Расчёт нормалей поверхности. Локальные модели освещения (база).
- Модели освещения тканей (дополнительно).
- Имитация микрорельефа (дополнительно).
- Тени. Карты теней (дополнительно).
- Редеринг в текстуру (дополнительно).
- Научная визуализация (дополнительно).

Предполагаемый объём кода задания: 1000 строк.

- Инициализация сетки 200 строк.
- Симуляция: 100 строк.
- Расчёт нормалей: 50 строк.
- Работа с OpenGL: 500-1000 строк.

Пример выполненного задания: https://youtu.be/q406SHgnJww

2 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. **НЕ нужно** включать в архив папку **ipch,** базы данных программы **.ncb, .sdf**.
    d. Проект должен собираться из папки src
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. Оборудование (процессор, **видеокарта**, объём памяти)
    d. Управление программой
    e. Реализованные пункты из бонусной части

3 База

Необходимо реализовать симуляцию и рендеринг полотна ткани.

3.1 Обязательные требования к симуляции:

- Движения ткани должны быть реалистичными.
- Движения должны затухать со временем.
- Должна быть возможность время от времени возобновлять движения.
  Либо автоматически при помощи ветра (как в примере) либо по кнопке 'p' (Push) добавлять возмущения.
- 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

3.2 Обязательные требования к визуализации

- В сцене должна быть текстурированная плоскость, имитирующая пол. Аналогично тому как сделано в примере. Либо какая-то более сложная имитация земли.
- В сцене должен присутствовать хотя бы 1 источник освещения: точечный или прожектор (spot). Просто 'Ambient' освещение не считается источником света.
- Один и тот же источник (или их совокупность) должен освещать и ткань, и плоскость.
- Ткань должна иметь текстуру отличную от текстуры плоскости.
- По кнопке '1' должен включаться каркасный режим визуализации полотна. Каждая линия должна отображать соединение между вершинами пружинкой.
- По кнопке '2' должна включаться визуализация нормалей в мировом пространстве цветом (база). Либо стрелочками (дополнительно).
- По кнопкам '3'-'6' должны включаться различные реалистичные варианты визуализации материала ткани. Для базы достаточно реализовать только 1 варинт (для базы достаточно Лабмерта с текстурой), по кнопке '3'.

4 Критерии оценки и дополнительная часть

Максимальная оценка за задание – **15 баллов**.

1. ***База.*** Симуляция полотна ткани и каркасная визуализация. Визуализация с текстурами в соответствии с требованиями (**8 баллов**).

2. ***Дополнительно*** (**7 балов максимум**, в каждом пункте **указано максимальное число баллов** за качественную реализацию этого пункта):

   - Реалистичная симуляция

     - Оригинальная форма ткани или моделируемый объект (**+1**). (флаг, одежда, шатёр/палатка, любые другие тканевые объекты)
     - Учёт столкновений полотна ткани с простыми объектами (сфера, бокс) (**+2**). Объекты необходимо визуализировать.
     - Анимация одежды на танцующем манекене (столкновения при помощи множества боксов и сфер) (**+6**). Не складывается с двумя предыдущими пунктами.
     - Реализация падения со сползанием такни по сфере или боксу (**+2**). По кнопке 'd' (Drop).
     - Синхронизация танцующего манекена с музыкой (**+2**). Пример синхронизации находится в примере 'advanced/gs_sound'. https://github.com/FROL256/msu-opengl4-sdk

   - Симуляция на GPU (OpenGL или OpenCL) (**+6**) Должна быть реализована возможность увеличения и уменьшения числа симулируемых и отрисовываемых объектов по кнопкам '+' и '-'. Должен быть реализован вывод времени симуляции куда-нибудь. Можно в название окна.

Кнопка '9' должна включать/отключать параллельную реализацию. **Скорость симуляции не должна зависеть от числа объектов! Даже при снижении FPS ниже 60 кадров в секунду.**

- +4 балла за расчёт через OpenGL Compute Shaders или OpenCL
- +6 баллов за расчёт через вершинный шейдер и transform fedeedback.
  В этом и только в этом случае нормали могут быть не сглажены! Вам необходимо рассчитать их в геометрическом шейдере.

- Симуляция на CPU параллельно (**+2**). Должна быть реализована возможность увеличения и уменьшения числа симулируемых и отрисовываемых объектов по кнопкам '+' и '-'. Должен быть реализован вывод времени симуляции куда-нибудь. Можно в название окна. Кнопка '9' должна включать/отключать параллельную реализацию. **Скорость симуляции не должна зависеть от числа объектов! Даже при снижении FPS ниже 60 кадров в секунду.**

- Тени
  - Тени на плоскости (**+1**)
  - Тени на плоскости и других объектах (**+3**) (простой Shadow Map)
  - Простой Shadow Map + PCF (**+4**)
  - Более сложные и реалистичные методы теней (**+6**)

  Внимание! Требования к реализации карт теней:
  - Если реализованы тени при помощи карт теней, 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 на этот объект. Аналогично тому как сделано в примере. Либо делать не плоскую землю.

- Реалистичные модели материала ткани.
  - Локальные модели (**+2**) в зависимости от реалистичности.
    Например, Oren–Nayar [3].
  - Имитация BSSRDF (**+3**) в зависимости от реалистичности. Например texture space diffusion [4].
  - Имитация золотых или серебрянных прожилок при помощи specular текстур (**+1**).
    (Подсказка: Для более реалистичной имитации бликов можно добавлять в сцену точечные источники, действующие только на specular).

- Имитация микрорельефа (normal mapping) (**+3**).
  - Обязателен корректный расчёт tangent space.

- Прозрачная ткань (**+1**) (как доп. реалистичный режим по кнопке '4' или '5')

- Цветная тень от прозрачной ткани (**+1**) (как доп. реалистичный режим по кнопке '4' или '5')

- Реалистичное окружение
  - Sky box с небом (**+1**)
  - Визуализация объекта, к которому крепится ткань (**+1**).
  - Добавление других произвольных мешей в сцену (**+2**)

- Научная визуализация
  - Визуализация сил, действующих на вершины сетки в виде стрелочек (**+3**) 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.

Задание из примера получило бы 10 баллов за базу, +1 балл за имитацию вставок при помощи отдельной текстуры, и +4 балла за карты теней с использованием PCF. Итого 15 баллов.

5 Теоретические основы

5.1 Физическая Модель

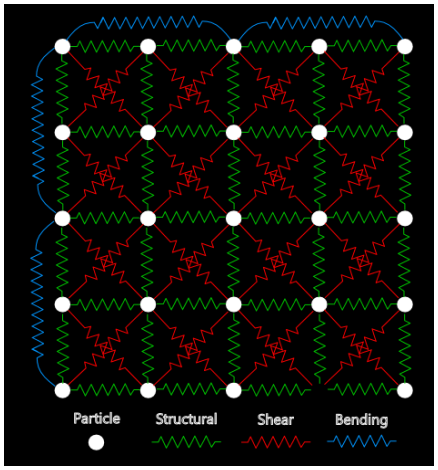

Рисунок 3. Простейшая модель полотна ткани

Модель ткани - это набор вершин, имеющих массу и соединённых пружинами. Пружинами соединяются смежные вершины (красные и зелёные) и "через одну" (синие). Разным типам пружин можно задать разную жёсткость.

Обратите внимание, что в модели для придания ткани упруости на изгиб присутствуют не только соединения смежных вершин, но и соединения “через 1” (синие пружинки). В тестовой реализации задания мы использовали модель с бОльшим количеством пружин, где каждая вершина соединяется со всеми смежными и всеми вершинами "через один" (Рисунок 4). Алгоритм построения такой модели дан ниже (алгоритм 1).

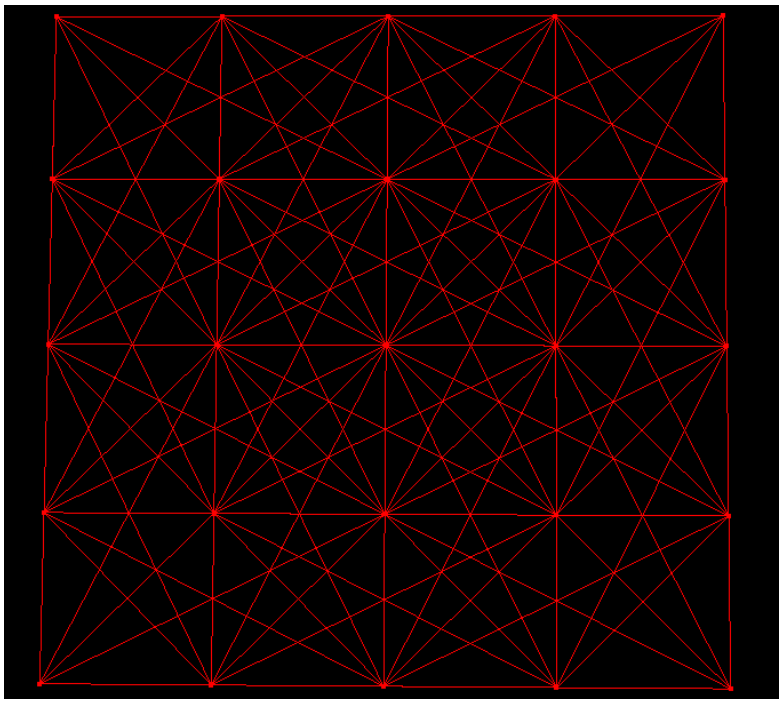

Рисунок 4. Соединения смежных вершин пружинами и "через одну". Разным типам пружин можно задать разную жёсткость.

```
std::vector<int>    edgeIndices;
std::vector<float>  edgeHardness;
std::vector<float>  edgeInitialLen;


for (size_t i = 0; i < vertPos.size(); i++)
{
  for (int j = i + 1; j < vertPos.size(); j++)
  {
    float4 vA = vertPos[i];
    float4 vB = vertPos[j];

    float dist = dist(vA, vB);
```

```
      if (dist < 2.0f*sqrtf(2.0f)*edgeLength)
      {
        float hardness = initialHardnes * (edgeLength / dist);

        edgeIndices.push_back(i);
        edgeIndices.push_back(j);
        edgeHardness.push_back(hardness);
        edgeInitialLen.push_back(dist);
      }
    }
  }
```

Алгоритм 1. Алгоритм задания соединений для вершин.

Алгоритм построения соединений "через 1" и смежных. Метод грубой силы, прост в реализации. В нашей реализации жёсткость пружины задавалась обратно пропорционально её длине, т.е. удельная жёсткость всех пружин одинакова. В задании вам не обязательно реализовывать именно такую модель. Возможны вариации числа соединений и их жёсткости. Например, вы можете использовать триангуляцию Делоне.

5.2 Основы симуляции и метода конечных элементов

Метод конечных элементов (*МКЭ*) — это численный метод решения дифференциальных уравнений с частными производными, а также интегральных уравнений, возникающих при решении задач прикладной физики [1]. Уравнения, описывающие движение системы из точек, соединённых пружинами, можно найти в [2]. Мы не будем подробно останавливаться на математической основе. Отметим лишь, что рассматриваемая далее модель эквивалентна методу конечных элементов первого порядка. Разобьём ось времени на набор достаточного большого числа маленьких интервалов.

Будем рассматривать поведение системы в моменты времени t0, t1, t2 (в центре интервала) и.т.д с небольшим шагом по времени Δt. Время Δt называется шагом дискретизации. В каждый момент времени точки (вершины), обладающие заданной массой, находятся в известной позиции. Они имеют определённые скорости. При этом, на каждую точку действуют силы упругости смежных пружин, определяемые их текущим растяжением/сжатием. Также действуют внешние силы: гравитация и ветер. Условие небольшого шага по времени позволяют в простейшей модели считать все указанные параметры константными в пределах от t[i] до t[i+1]

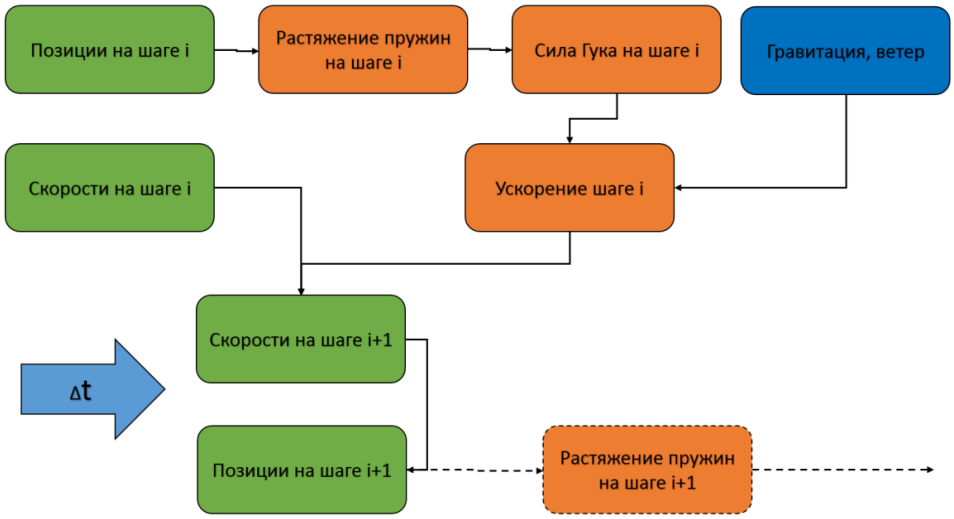

Рисунок 6. Схема моделирования.

На каждом шаге вам необходимо хранить только позиции и скорости вершин. Зная позиции на текущем шаге, можно вычислить текущие длины всех пружин. Зная текущую длину пружины, можно по закону Гука вычислить силу упругости, действующую на вершины на двух её концах. Путём векторного сложения всех сил, действующих на данную вершину, можно получить результирующую силу упругого воздействия пружин. Добавив ветер и гравитацию, получаем результирующую силу для данной вершины на текущем шаге. Зная её, можно вычислить ускорение. Зная ускорение и текущие скорости, можно вычислить новые скорости - на следующем шаге. Наконец, зная скорости, можно вычислить новые позиции и переходить к следующему шагу по времени.

5.3 Основные формулы

$$\overrightarrow{F_{\text{упр}}} = \vec{n}k\Delta l$$

где $\vec{n}$ – единичный вектор направления пружины, k – коэффициент упругости, Δl – величина растяжения-сжатия пружины со знаком

$$\overrightarrow{F_{\text{рез.}}} = \sum \overrightarrow{F_{\text{упр}}} + \overrightarrow{F_{\text{ветра}}} + \overrightarrow{F_{\text{тяжести}}}$$

$$\vec{a} = \frac{\vec{F}}{m}$$

$$\Delta\vec{V} = \vec{a}\Delta t$$

$$\overrightarrow{\Delta Pos} = \overrightarrow{V_0}\Delta t + \vec{a}\frac{\Delta t^2}{2}$$

Внимание, хак!

Мы заметили, что вычисления позиций на новом шаге можно производить по другой формуле, неправильной с точки зрения МКЭ первого порядка и с физической точки зрения, но дающей визуально более гладкий результат.

Неправильная с точки зрения МКЭ 1 порядка (но работающая) формула:

$$\overrightarrow{\Delta Pos} = \left(\overrightarrow{V_0} + \Delta\vec{V}\right)\Delta t + \frac{\vec{a}\Delta t^2}{2}$$

Поскольку целью задания не является физически точная симуляция, вы можете использовать эту или любые другие аппроксимации, делающие движения ткани более плавными и реалистичными. **Однако будьте аккуратны, любые хаки, как правило, делают симуляцию менее устойчивой!!!**

5.4 Основы геометрических преобразований и OpenGL 3/4

(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.

Важная подсказка по реализации отображения треугольного меша и меша из линий:

(4) Вы можете создать дополнительный буфер индексов и дополнительный VAO, чтобы рисовать тот же самый меш не только линиями, но и треугольниками. Не нужно дублировать буферы, содержащие атрибуты вершин.

5.5 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 [3].

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.

5.5.1 Вычисление сглаженных нормалей

Важным требованием базовой части задания является расчёт сглаженных нормалей. Это необходимо для того чтобы получить плавно меняющееся освещение по поверхности ткани. Иначе будут видны отдельные грани. Для того чтобы получить сглаженные нормали в вершинах, вам сначала необходимо вычислить нормали к квадам (или треугольникам) при помощи векторного произведения, а затем для каждой вершины усреднить нормали всех квадов (или треугольников), которые её касаются.

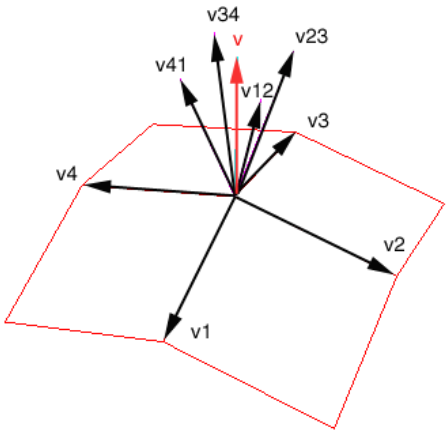

Рисунок 7. Иллюстрация вычисления сглаженных нормалей в вершинах. Нормали к полигонам усредняются, чтобы получить нормаль в вершине.

5.6 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 8 и 9).

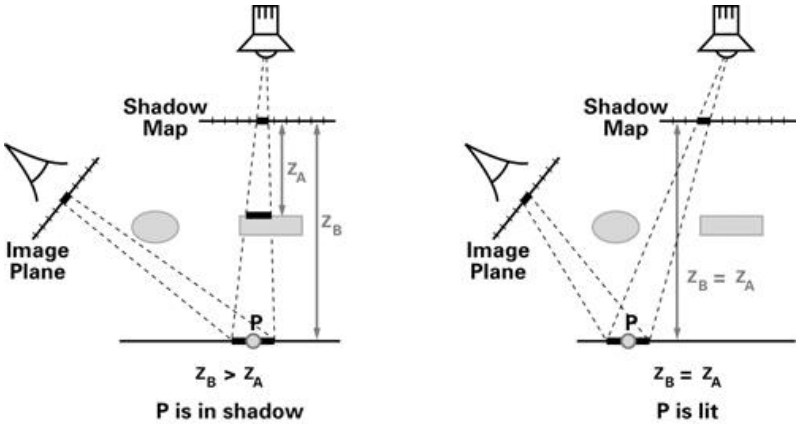

Рисунок 8. Иллюстрация работы метода карт теней.

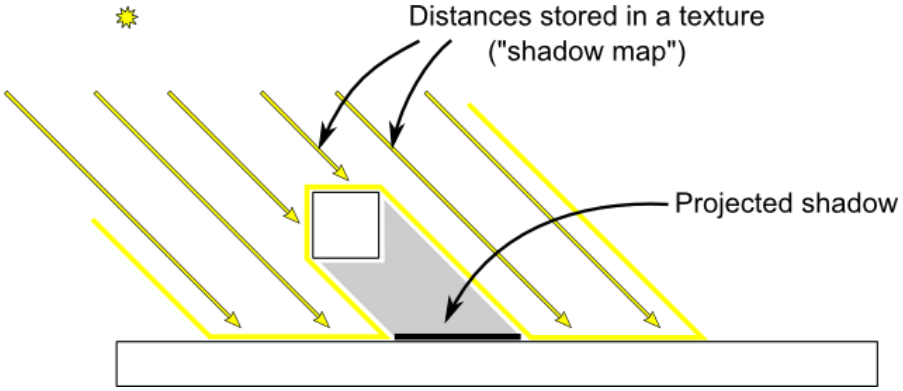

Рисунок 9. Ещё одна иллюстрация работы метода карт теней.

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. Можно использовать предоставляемую библиотечную функцию `glusLookAtf`.
2. Расчёт матрицы проекции для вида из источника освещения. Можно использовать предоставляемую библиотечную функцию `glusOrthof` или `glusPerspectivef`.

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 примирения матрицы проекции на 4 шаге вы должны поделить получившиеся координаты на w (алгоритм 2).

```
vec4 posLightSpace     = shadowViewMatrix*vec4(fragmentWorldPos, 1);
vec4 posLightClipSpace = shadowProjMatrix*posLightSpace;
vec2 shadowTexCoord    = (posLightClipSpace.xy/posLightClipSpace.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.

5.7 Рендеринг в текстуру в OpenGL

Рендеринг в текстуру в OpenGL реализуется при помощи абстракции, называемой Frame Buffer Object (FBO) (рис 9).

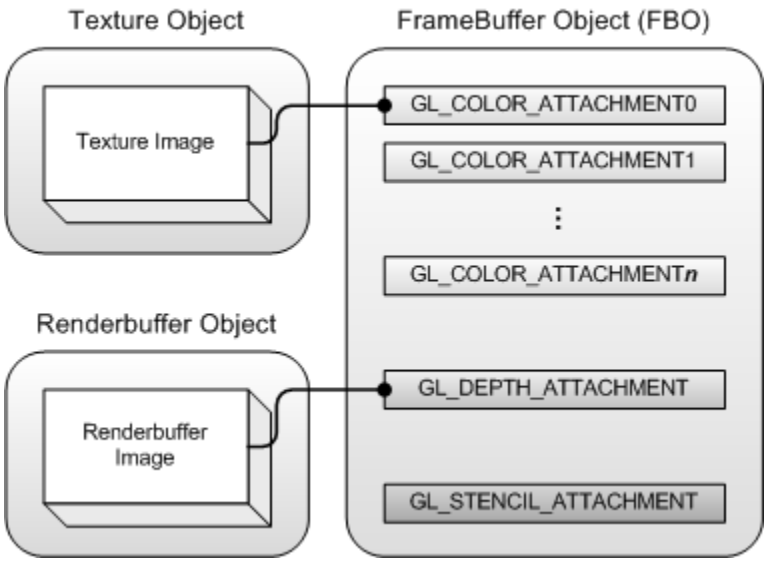

Рисунок 9. Иллюстрация работы Frame Buffer Object (FBO). Хотя на данном рисунке к GL_DEPTH_ATTACHMENT прикреплён специальный объект renderbuffer, вместо него может быть текстура специального формата. Именно так сделано в предоставляемом вам шаблоне.

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.

5.8 Подсказки о симуляции на GPU

Внимание! Симуляция на GPU – это достаточно сложная задача. Прежде всего, потому что на одну вершину действует сразу несколько пружин. А поскольку на GPU все операции записи в буферы позиций и скоростей будут проходить параллельно, нельзя просто распараллелить работу по рёбрам (пружинам). Существует 2 варианта решения этой проблемы:

1. Рассчитать силы для каждой пружины, и суммировать силы в каждой вершине при помощи атомарных операций. Но это медленно! (хотя может быть быстрее, чем на CPU)

2. Распараллеливать вычисления по вершинам и для каждой вершины на GPU передавать списки соединённых с ней пружинами других вершин. Эти списки можно составить, например, заранее и передать на GPU в виде буфера или текстуры с целочисленным форматом.

Для выполнения симуляции на GPU вы можете использовать 3 механизма:

1. Расчёт позиций и скоростей в вершинном шейдере и сохранение позиций и скоростей при помощи transform feedback. Этот механизм хорош тем, что он имеет поддержку на достаточно большом числе GPU и минимизирует объём как вычислений, так и нагрузки на шину памяти, поскольку вычисленные позиции вершин сразу отправляются дальше по графическому конвейеру, формируя треугольники и создавая дополнительную работу для фрагментного шейдера. Этот способ самый эффективный и самый правильный с точки зрения индустрии gamedev-а.

   При реализации симуляции этим способом вам придётся вычислять нормали не по-вершинно а по треугольникам в геометрическом шейдере. В этом и только в этом случае допускаются несглаженные нормали.

2. Расчёт позиций и скоростей в OpenGL Compute Shaders (4.3 и выше). Этот путь немного проще, чем предыдущий, и менее эффективен, т.к. при последующем запуске на отрисовку вершинный шейдер выполнит одну лишнюю операцию чтения.

Расчёт нормалей должен быть выполнен как дополнительный вызов compute shader-а, вычисляющий сглаженные по-вершинные нормали.

3. Расчёт позиций и скоростей в OpenCL и использования механизма взаимодействия с OpenGL чтобы OpenCL мог напрямую записывать данные в буфера, которые будут доступны для OpenGL.

   С нормалями аналогично пункту 2. Должно быть отдельное ядро, рассчитывающее нормали.

6 Литература

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

[1] Коэффициент Упругости. Закон Гука. Информация из википедии. https://ru.wikipedia.org/wiki/Коэффициент_упругости

[2] Метод конечных элементов. Доц. Хандримайлов ХНАДУ. Кафедра теоретической механики и гидравлики. URL = http://files.khadi.kharkov.ua/faily/item/download/5176_5d6ce64a0297a2f261d20213726a01a8.html

[3] Боресков А. В. Модели Освещения. URL = http://steps3d.narod.ru/tutorials/lighting-tutorial.html

[4] Texture Space Diffusion. http://http.developer.nvidia.com/GPUGems/gpugems_ch16.html

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

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

[7] 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

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

[9] 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] FIDESYS. Отечественный промышленный пакет прочностного анализа, основанный на МКЭ. URL = http://www.cae-fidesys.com/ru.