**Лабораторная работа 4.**

**«Шейдеры»**

1. Задание

Знакомство с современными технологиями вывода аппаратно ускоренной трехмерной графики.

Создать программу, рисующую сцену используя шейдеры.

Сцена содержит объекты, покрытые несколькими текстурами, полупрозрацные объекты и источники света.

Реализовать несколько интересных моделей освещения.

1. Теория

Для современных графических процессоров шейдеры - это некоторая программа, выполняемая непосредственно графическим процессором. GPU работает и с данными вершин и с отдельными фрагментами изображения. Поэтому выделяют 2 самых известных типа шейдеров: вершинные и фрагментные.

Вершинный шейдер -- это выполняемая графическим процессором программа, обрабатывающая данные вершин геометрических примитивов.

К таким данным относятся координаты вершины в пространстве, текстурные координаты, вектор нормали и др. Вершинный шейдер может быть использован для видового и перспективного преобразования вершин, генерации текстурных координат, расчета освещения и т. д.

Он получает все параметры вершины, может по своему усмотрению изменить некоторые из них, рассчитать на основе полученных параметров вспомогательные значения. Также в его непосредственные обязанности входит непосредственное проецирование 3-мерной вершины на 2-мерную плоскость экрана.

Вершинный шейдер не обладает доступом к топологической информации (например: организация отдельных вершин в геометрические примитивы) и не может изменять существующую топологию. Вершинный шейдер не имеет доступа данным к другим вершинам, и обычно это связано с тем, что вершины на GPU обрабатываются параллельно и независимо друг от друга.

Фрагментный (пиксельный) шейдер -- это выполняемая графическим процессором программа обработки отдельных фрагментов, получаемых при растеризации геометрических примитивов. Т.е. фрагментный шейдер работает с фрагментами изображения, а под фрагментом изображения в данном случае понимается пиксель, которому поставлен в соответствие некоторый набор атрибутов, таких как цвет, глубина, текстурные координаты.

Фрагментные шейдеры также выполняются параллельно для нескольких фрагментов.

В настоящее время выделяется еще один тип шейдеров -- геометрический.

Геометрический шейдер, в отличие от вершинного, способен обработать не только одну вершину, но и целый примитив. Это может быть отрезок (две вершины) и треугольник (три вершины), а при наличии информации о смежных вершинах может быть обработано до шести вершин для треугольного примитива. Кроме того геометрический шейдер способен генерировать примитивы «на лету», не задействуя при этом центральный процессор.

Он получает все необходимые параметры, такие как координаты в текстуре, примешиваемый цвет и другие. В его непосредственные обязанности входит вернуть цвет пикселя.

В чем основное преимущество шейдеров?

Средствами обычного OpenGL можно сделать освещение Ламберта или Гуро -- стандартные методы освещения. Однако освещение Фонга или освещение с использованием карт нормалей (и другие и нтересные модели освещения) обычными средствами OpenGl уже не сделаешь. И тут уже не обойтись без шейдеров.

Также, при помощи шейдеров можно делать неограниченное количество источников света(количество источников уже будет зависеть от мощности видеокарты), в то время как OpenGl допускает всего 8 источников.

1. Алгоритм

Рассмотрим подробнее используемое освещение:

*1) освещение Блинна с бликами*

Бликовая модель Блинна - одна из наиболее распространенных моделей.

Освещенность Блинна задается по следующей формуле (стр 280 Боресков)

Расшифровка параметров:

ka -- диффузное освещение

kd -- фоновое освещение

n -- единичный вектор нормали к поверхности

l -- единичный вектор на источник света

ks -- параметр бликового освещения

I -- цвет источника света

h -- единичный вектор - бисектор направления на источник света l и направления на наблюдателя v.

p -- контрастность блика -- чем он больше, тем более контрастным (и меньшего размера) получается блик.

Формула пригодна для расчета освещенности в каждом пикселе. Из этого следует, что нам необходимо знать единичные векторы n и l для каждой вершины и передавать мы их будем во фрагментный шейдер с помощью varying-переменных.

Через uniform-переменные в вершинный шейдер передаются координаты источника света. А во фрагментном шейдере мы для каждого фрагмента получим значения векторов n, l, v и h, полученных путем интерполяции значений, найденных в вершинном шейдере.

Бликовое освещение меняется в соответствии с вращением солнца из лабораторной работы №2.

*2) освещение Фонга с текстурированием*

Еще одна бликовая модель освещения, которая использует формулу:

Все параметры аналогичны перечисленным выше, за исключением r:

r -- отраженный относительно вектора n вектор на наблюдателя v

В формуле С -- цвет поверхности. В обычной модели Фонга С -- константа. Однако, для текстуированной версии С -- цвет, который получается при обращении к текстуре.

Код вершинного шейдера (sea\_frag.glsl):

//~ освещение Фонга с текструированием

varying vec3 n;

varying vec3 l1;

varying vec3 v1;

uniform sampler2D decalMap;

vec4 setLightTexFong() {

const vec4 specColor = vec4( 0.7, 0.7, 0.0, 1.0);

const float specPower = 20.0;

vec4 diffColor = texture2D(decalMap, gl\_TexCoord[0].xy);

vec3 n2 = normalize(n);

vec3 l2 = normalize(l1);

vec3 v2 = normalize(v1);

vec3 r = reflect(-v2,n2);

vec4 diff = diffColor \* max( dot(n2,l2), 0.0);

vec4 spec = specColor \* pow( max(dot(l2,r),0.0), specPower);

return diff + spec;

}

//~ blinn

varying vec3 lb;

varying vec3 hb;

varying vec3 vb;

vec4 setLightBlinn() {

const vec4 diffColor = vec4 ( 0.7, 0.3, 0.5, 1.0);

const vec4 specColor = vec4 ( 0.7, 0.7, 0.0, 1.0);

const float specPower = 20.0;

vec3 n2 = normalize(n);

vec3 l2 = normalize(lb);

vec3 h2 = normalize(hb);

vec4 diff = diffColor \* max ( dot(n2,l2), 0.0);

vec4 spec = specColor \* pow ( max (dot(n2,h2),0.0),specPower);

return diff+spec;

}

void main() {

vec4 color = texture2D(decalMap,gl\_TexCoord[0].st);

gl\_FragColor = (setLightTexFong()+setLightBlinn())\*color;

}

Код пиксельного шейдера (sea\_vert.glsl):

//~ освещение Фонга с текструированием

varying vec3 n;

varying vec3 l1;

varying vec3 v1;

uniform vec4 lightPos;

uniform vec4 eyePos;

void setLightTexFong(){

vec3 p = vec3( gl\_ModelViewMatrix \* gl\_Vertex);

l1 = normalize( vec3(lightPos) - p);

v1 = normalize( vec3(eyePos) - p);

n = normalize( gl\_NormalMatrix \* gl\_Normal);

}

varying vec3 lb;

varying vec3 hb;

varying vec3 vb;

uniform vec4 lightPosb;

uniform vec4 eyePosb;

void setLightBlinn() {

vec3 p = vec3 ( gl\_ModelViewMatrix \* gl\_Vertex);

lb = normalize ( vec3(lightPosb) - p);

vb = normalize ( vec3(eyePosb) - p);

hb = normalize ( lb+vb);

n = normalize ( gl\_NormalMatrix \* gl\_Normal);

}

void main() {

setLightTexFong();

setLightBlinn();

gl\_Position = gl\_ModelViewProjectionMatrix \* gl\_Vertex;

gl\_TexCoord[0] = gl\_MultiTexCoord0;

}

В вершинном шейдере через переменную gl\_TexCoord[0] передаются текстурные координаты для фрагментного шейдера. Без этого во фрагментном шейдере у всех фрагментов было бы одно и тоже значение текстурных координат.

1. Выводы

С помощью шейдеров можно создавать свои собственные модели освещения, программировать сложные физические эффекты для конкретных моделей и материалов.

Например, освещение с помощью модели Фонга или добавление эффекта стеклянного объекта, для имитации неровностей и др.

Часто шейдеры используются для реализации эффекта postprocessing. Смысл этого процесса в том, что необходимый эффект производится после рендеринга сцены.

Также, шейдеры используются для увеличения быстродействия. Так как пропускная способность шины, через которую идут данные на видеокарту, не так велика, как хотелось бы, то возникает идея: имеет смысл как можно больше данных держать на ивдеокарте и как можно меньше отправлять каждый новый кадр на видеокарту. Основные примеры оптимизации с использованием шейдеров:

1) системы частиц

2) морфинг

3) скелетная анимация

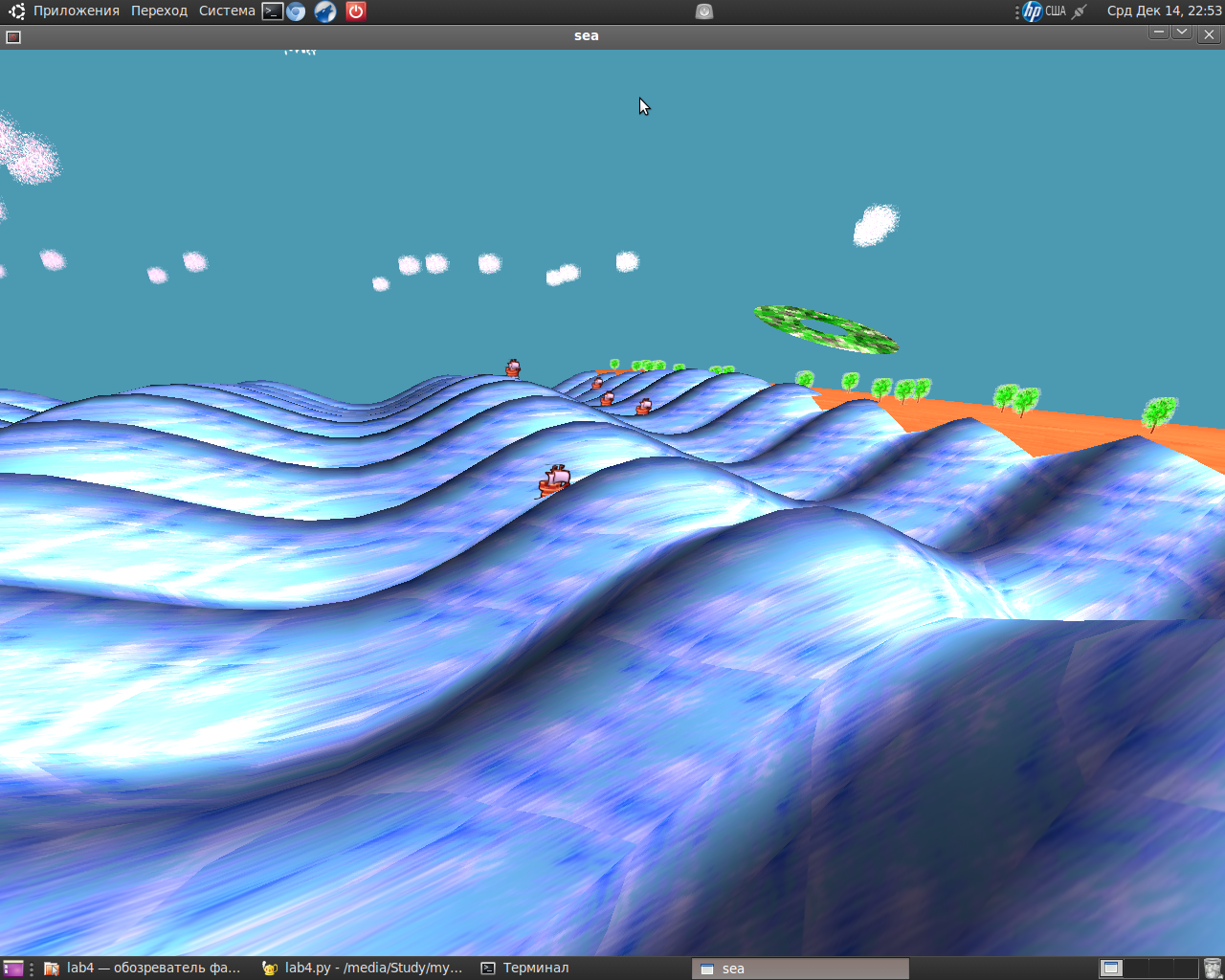
4) любые искажения объектов

Нужно помнить и о том, что применение сложных шейдеров может замедлить работу программы.

Основным ограничением вершинных шейдеров является то, что каждая вершина обрабатывается не имея никакой информации об остальных вершинах, в первую очередь вершинах, образующих примитив, содержащий данную вершину. Но именно это позволяет легко распараллеливать обработку вершин и получить высокую скорость рендеринга.

1. Литература
2. OpenGL Shading Language (Orange Book)
3. Боресков А.В. Разработка и отладка шейдеров.
4. Сморкалов Андрей. «OpenGL. Шейдеры»

**Screen:**

****

