МИНИСТЕРСТВО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ
АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ИНСТИТУТ)

Лабораторная работа 5

Работа с текстурной памятью

Студент: *Аксенов С.Ю.*
Преподаватель: *Семенов С.А.*
Оценка:

Москва, 2012

Содержание

Постановка задачи

Разработать программу выполняющую задание определённое вариантом. Программа должна задействовать возможности графического процессора посредством CUDA API. Также, прграмма должна выполняться параллельно, распределять память и использовать векторные операции. Программа должна выполнять проверки на ошибки, в частности проверять наличие аппаратной поддержки CUDA. Также программа задействует библиотеку MS-MPI для вычисления нескольких независимых блоков модели.

Вариант 1. Трассировка лучей. Отражения

Алгоритм

Задаются размеры пиксельной сетки, определяется положение наблюдателя, его ориентация в пространстве, плоскость просмотра, угол обзора. Затем задаются объекты сцены (в данном случае сферы). Далее в соответствии с сеткой из точки местонахождения наблюдателя испускаются лучи проходящие через центры пикселов.

При прохождении луча по сцене проверяется, не проходит-ли он через каки-либо её объекты. Если да, то в соответствии со свойствами материала объекта производятся дальнейшие расчёты возникающих новых лучей

Реализация

Главным в данной программе является трассировщик лучей, поэтому его и рассмотрим:

```
1  __device__ HitInfo intersect(Ray &ray,
2    unsigned char objectsSize, Sphere *ignore)
3    {
4      HitInfo result = HitInfo(FLT_MAX, NULL);
5      float dist;
6
7      for (unsigned char i = 0; i < objectsSize; i++)
8      {
9        if (objects+i != ignore)
10       {
11         dist = tex1Dfetch(i).hit(ray);
12         if (dist < result.hitTime && dist > 0)
13         {
14           result.hitTime = dist;
15           result.hitObject = objects+i;
16         }
17       }
18     }
19
20     if (result.hitObject != NULL)
```

```
21      {
22        result.hitPoint = ray.getPos(result.hitTime);
23        result.normal = result.hitObject->normal(result.hitPoint);
24      }
25      return result;
26    }
27
28  __device__ Vector3D shade(Ray &ray, unsigned char objectsSize,
29    Sphere *ignore, Sphere *currObj)
30    {
31      __shared__ Vector3D background, ambient;
32      Vector3D color;
33      Vector3D reflect;
34      Ray r;
35      background = Vector3D(0.8, 0.8, 1.0);
36      ambient = Vector3D(0.1, 0.2, 0.1);
37
38      HitInfo hitInfo = intersect(ray, objectsSize, ignore);
39      if (NULL == hitInfo.hitObject)
40      {
41        color = background;
42      }
43      else
44      {
45        r.start = hitInfo.hitPoint;
46
47        if (hitInfo.hitObject->data.Krefl > 0 &&
48        currObj != hitInfo.hitObject)
49        {
50          r.dir = ray.dir - 2 * hitInfo.normal * Vector3D::dotProduct(ray.dir
51          HitInfo hitInfo2 = intersect(r, objectsSize, hitInfo.hitObject);
52          if (NULL == hitInfo2.hitObject)
53          {
54            reflect = background;
55          }
56          else
57          {
58            reflect = hitInfo2.hitObject->data.Kamb * ambient;
59          }
60        }
61        else reflect = Vector3D();
62
63        color = hitInfo.hitObject->data.Kamb * ambient +
64        hitInfo.hitObject->data.Krefl * reflect;
65      }
66
```

```
67      return color;
68    }
69
70    __global__ void OnGPU(unsigned char objectsSize,
71    Vector3D *eyeD, Vector3D *vX, Vector3D *vY,
72    float W, float H, float step, float3 *result)
73    {
74      Vector3D eyePos = Vector3D(0, 0, 0);
75      Vector3D eyeDir = *eyeD;
76      Vector3D vx = *vX;
77      Vector3D vy = *vY;
78
79      float x = -W + blockIdx.y * step;
80      float y = -H + blockIdx.x * step;
81
82      Ray ray;
83      ray.start = eyePos;
84      ray.dir = Vector3D(eyeDir + vx * x + vy * y);
85      ray.dir.normalize();
86      Vector3D color = shade(ray, objects, objectsSize, NULL, NULL);
87  }
```

Функция *shade* выполняет «закраску» пикселя, по которому прошёл луч, функция *intersect* производит проверку на столкновение луча с объектами сцены.

Из недостатков предложенного решения можно отметить радикальный способ убрать рекурсию — код функции закраски дублируется вручную в коде отражения и выполняется только один раз.

Сравнение производительности GPU и CPU

Скомпилировано с помощью Visual Studio SP1, в конфигурации Release (Win32). Версия CUDA Toolkit — 4.2. Видеокарта — GT 520MX, центральный процессор — Intel Pentium B940, 2 GHz (2 cores). При размерах сетки 640 на 480 на CPU время выполнения составило 50 миллисекунд. На GPU — порядка 10 миллисекунд.

Выводы

Выполнение трассировки лучей на видеокарте имеет очень важное применение — оно позволяет добиться изображения гораздо лучшего качества, чем при обычных подходах, и при этом производить их с приемлемым fps, в реальном времени. Даже при «лобовых» алгоритмах.

Список используемой литературы

- Боресков, Харламов. Основы работы с технологией CUDA.