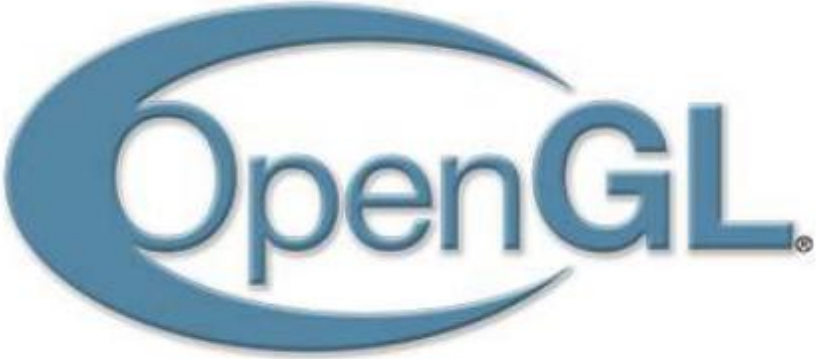
Что такое OpenGL / DirectX?

OpenGL и DirectX представляют из себя графический API, то есть набор функций для рисования сложных графических сцен.

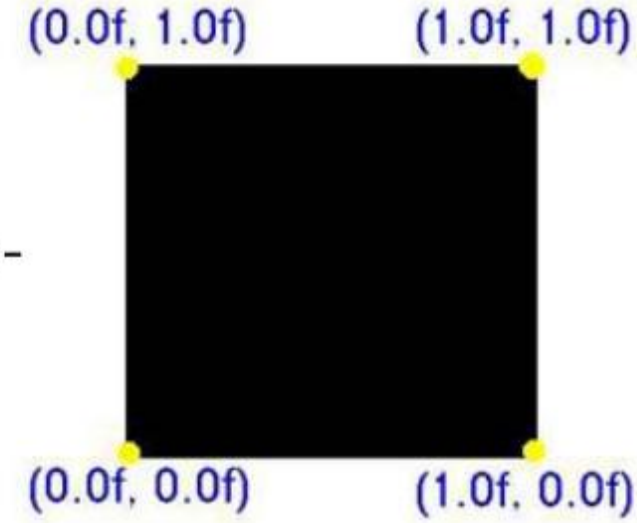


Microsoft®
DirectX®

Как представлен объект в OpenGL / DirectX?

- В основном, в этих API объекты представляют из себя набор точек, линий и треугольников, так как с помощью треугольников легко можно представить поверхность даже со сложными изломами.

- Каждой точке соответствует текстурная координата, что позволяет накладывать на объекты текстуры.



Для чего здесь CUDA?

- Обычно CUDA используется для параллельных рассчетов на GPU.

- Так почему бы не использовать GPU по его первоначальному назначению - рассчетов, связанных с визуализацией?

- Визуализация требует разнообразных тяжелых рассчетов, обычно они однотипные для каждого объекта (луч, треугольник и т.д.), что делает параллельные вычисления особенно востребованными.

- Особенно если это рассчеты в реальном времени, где из-за скорости может пострадать плавность картинки.

Задачи визуализации, где можно использовать CUDA

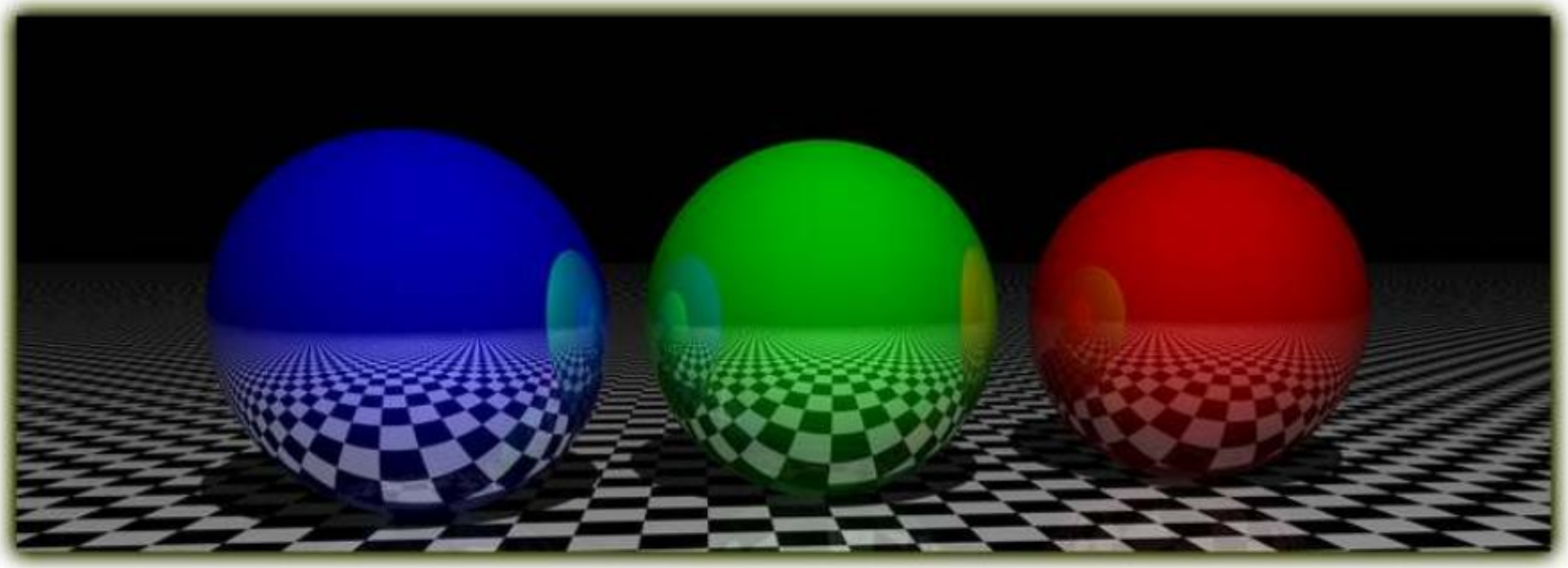
Симуляция системы частиц. Множество объектов (частиц), следующее состояние которых зависит от предыдущего состояния системы, легко распалеливается в связи с большим количеством однотипных вычислений.

Задачи визуализации, где можно использовать CUDA

Сложные рассчеты. Разнообразные сложные рассчеты, которые легко распараллелить, например, симуляция водной глади с помощью быстрого преобразования Фурье (FFT).

Задачи визуализации, где можно использовать CUDA

Трассировка лучей - технология отслеживания обратной траектории распространения луча (от экрана к источнику). Требует большого количества однотипных рассчетов пересечений (каждого луча с каждым объектом на сцене), распараллеливание просто напрашивается, особенно, если речь о реальном времени.



Задачи визуализации, где можно использовать CUDA

Постпроцессинг - технология обработки уже отрендеренной в текстуру сцены, с помощью каких-либо эффектов, которые можно применить к двумерному изображению.

Почему бы не использовать шейдеры? (GLSL/HLSL)

Шейдер - программа, выполняющаяся на GPU на одной из ступеней графического конвеера, то есть во время создания 2D или 3D сцены.

- Изначально шейдеры были придуманы для определения окончательного состояния объекта или изображения. И, чтобы написать на них параллельный код, придется представить все данные в виде текстур, а вычисления, как смешивание текстур (что усложняет задачу программисту).

- Кроме того, CUDA все равно работает быстрее, результаты по времени сравнимы, но CUDA впереди.

- Но есть и плюс - отсутствие строгой привязки к железу.

Как использовать CUDA вместе с OpenGL / DirectX?

- CUDA используется для рассчетов, OpenGL / DirectX для их визуализации на экране.

- CUDA использует привычную для C-программиста работу с памятью (malloc, работа с указателями), OGL / DirectX же используют для хранения данных абстрактные буферы.

- CUDA предоставляет механизм, позволяющий отобразить OGL / DirectX буферы данных в пространство памяти CUDA.

- Причем работает это быстрее любых, даже простейших вычислений, что позволяет пренебречь скоростью взаимодействия CUDA с OGL / DirectX.

CUDA + OpenGL (Vertex Array)

Шаг 1. *Выделение в памяти и регистрация буфера вершин.*

```
GLuint vertexBuffer;
glGenBuffers( 1, &vertexBuffer);
glBindBuffer( GL_ARRAY_BUFFER, vertexBuffer);
glBufferData( GL_ARRAY_BUFFER, numVertices * 16, NULL,
    GL_DYNAMIC_COPY );
cudaGLRegisterBufferObject( vertexBuffer );
```

Дорогостоящая операция, следует использовать один раз для каждого буфера.

CUDA + OpenGL (Vertex Array)

Шаг 2. *Создание / измнение данных с помощью CUDA*.

```
void* vertexPtr;
cudaGLMapBufferObject( &vertexPtr, vertexBuffer);
VerticiesKernel<<<gsz,bsz>>>( vertexPtr, numVerticies );
cudaGLUnmapbufferObject( vertexBuffer );
```

Map / Unmap буфера недорогая операция, можно использовать всякий раз при вызове ядра, которому нужен этот буфер.

CUDA + OpenGL (Vertex Array)

Шаг 3. *Отрисовка с помощью OpenGL.*

glBindBuffer(GL_ARRAY_BUFFER, vertexBuffer);
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_COLOR_ARRAY);
glVertexPointer(3, GL_FLOAT, 16, 0);
glColorPointer(4, GL_UNSIGNED_BYTE, 16, 12);
glDrawArrays(GL_POINTS, 0, numVerticies);

- *можно также использовать*: GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP, GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, GL_QUADS, GL_QUAD_STRIP, GL_POLYGON

glDisableClientState(GL_COLOR_ARRAY);
glDisableClientState(GL_VERTEX_ARRAY);

CUDA + OpenGL (Работа с изображением)

Шаг 1. *Выделение и регистрация буфера пикселей.*

```
GLuint bufferID;
glGenBuffers( 1, &bufferID );
glBindBuffer( GL_PIXEL_UNPACK_BUFFER, bufferID );
glBufferData( GL_PIXEL_UNPACK_BUFFER, Width * Height * 4,
    NULL, GL_DYNAMIC_COPY );
cudaGLRegisterBufferObject( bufferID );
```

Как и в случае с вершинным буфером, дорогостоящая операция, следует использовать один раз для каждого буфера.

CUDA + OpenGL (Работа с изображением)

Шаг 2. *Создание текстуры*.

```
GLuint textureID;
glEnable( GL_TEXTURE_2D );
glGenTextures( 1, &textureID );
glBindTexture( GL_TEXTURE_2D, textureID );
glTexImage2D( GL_TEXTURE_2D, 0, GL_RGBA8, Width, Height,
    0, GL_BGRA, GL_UNSIGNED_BYTE, NULL );
```

Последний параметр NULL, так как нам нужно лишь выделить память под текстуру без инициализации.

CUDA + OpenGL (Работа с изображением)

Шаг 3. *Создание / изменение данных буфера с помощью CUDA*.

```
void* texturePtr;
cudaGLMapBufferObject( &texturePtr, bufferID);
TextureKernel<<<gsz,bsz>>>( texturePtr, numVerticies );
cudaGLUnmapbufferObject( bufferID );
```

Шаг 4. *Получение текстуры из буфера*.

```
glBindBuffer( GL_PIXEL_UNPACK_BUFFER, bufferID );
glBindTexture( GL_TEXTURE_2D, textureID );
glTexSubImage2D( GL_TEXTURE_2D, 0, 0, 0, Width, Height,
    GL_BGRA, GL_UNSIGNED_BYTE, NULL);
```

Последний параметр (source) NULL указывает на то, что данные будут получены из текущего буфера.

CUDA + OpenGL (Работа с изображением)

Шаг 5. *Рисуем текстуру на квадрате для вывода на экран.*

```
glBegin( GL_QUADS );
    glTexCoord2f( 0, 1.0f );
    glVertex3f( 0, 0, 0 );
    glTexCoord2f( 0, 0 );
    glVertex3f( 0,1.0f, 0 );
    glTexCoord2f( 1.0f, 0 );
    glVertex3f( 1.0f, 1.0f, 0 );
    glTexCoord2f( 1.0f, 1.0f );
    glVertex3f( 1.0f, 0, 0 );
glEnd();
```

Указывая каждой вершине соответствующую текстурную координату.