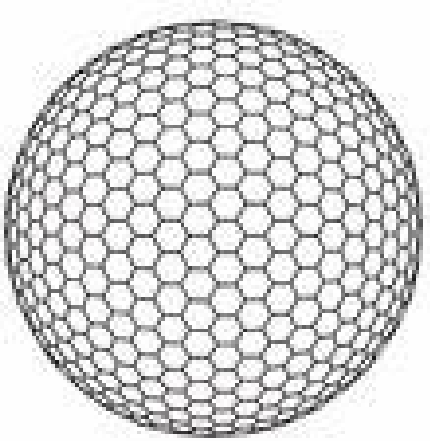
Текстуры в CUDA

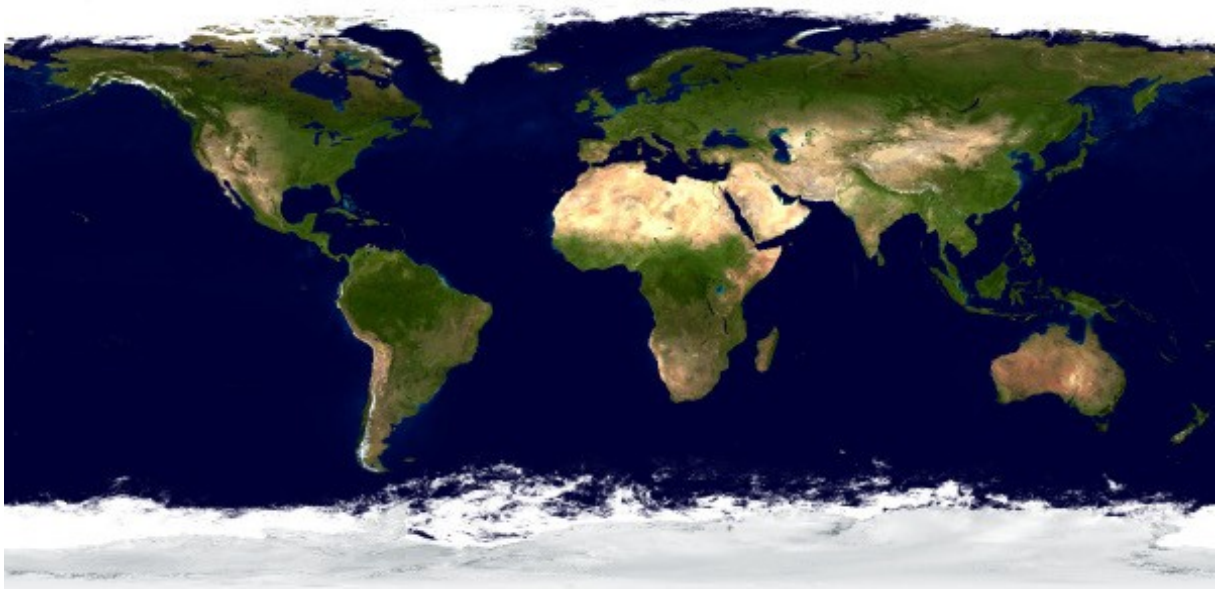
Романенко А.А.
arom@ccfit.nsu.ru

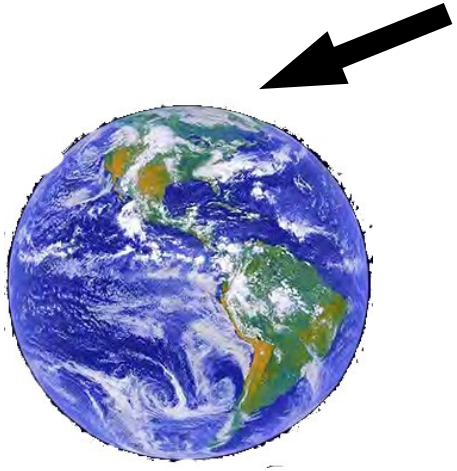
Что такое текстура?

- Способ доступа к данным



+





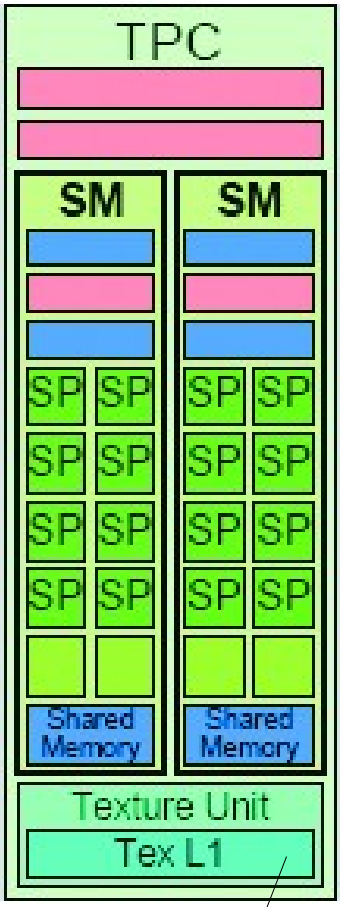
Особенности текстур

- Латентность больше, чем у прямого обращения в память
  - Дополнительные стадии в конвейере:
    - Преобразование адресов
    - Фильтрация
    - Преобразование данных
- Но зато есть кэш
- Разумно использовать, если:
  - Объем данных не влезает в shared память
  - Паттерн доступа хаотичный
  - Данные переиспользуются разными потоками

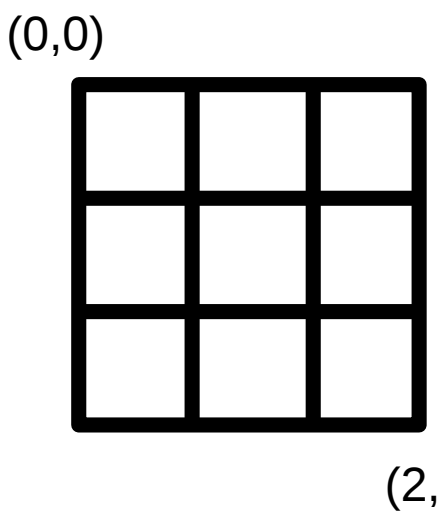
Свойства текстур

- Доступ к данным через кэш
  - Оптимизированы для доступа к данным которые расположены рядом в двумерном пространстве
- Фильтрация
  - Линейная/квадратичная/кубическая
- Свертывание (выход за границы)
  - Повторение/ближайшая граница
- Адресация в 1D, 2D и 3D
  - Целые/нормализованные координаты

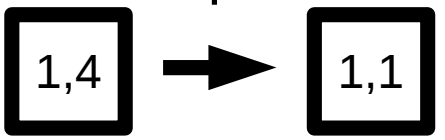
Свойства в картинках



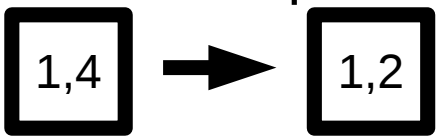
Кэш текстуры



Повторение



Ближайшая граница



- tex1Dfetch(texRef, x)
- tex1D(texRef, x)
- tex2D(texRef, x, y)
- tex3D(texRef, x, y, z)

Свойства в картинках

- Нормализация координат
- Фильтрация

Свойства текстур

- Преобразование данных:
  - **cudaReadModeNormalizedFloat** :
    - Исходный массив содержит данные в integer,
    - возвращаемое значение во **floating point** представлении (доступный диапазон значений отображается в интервал [0, 1] или [-1,1])
  - **cudaReadModeElementType**
    - Возвращаемое значение то же, что и во внутреннем представлении

Типы/свойства текстур

- Привязанные к линейной памяти
  - Только 1D
  - Целочисленная адресация
  - Фильтры и свертывание отсутствуют
- Привязанные к массивам CUDA
  - 1D, 2D или 3D
  - целые/нормализованные координаты
  - Фильтрация
  - Свертывание

Работа с текстурами

- Host:
  - Выделить память (malloc/cudaMallocArray/...)
  - Объявить указатель на текстуру
  - Связать указатель на текстуру с областью памяти
  - После использования:
    - Отвязать текстуру, освободить память
- Device:
  - Чтение данных через указатель текстуры
  - Текстуры для линейной памяти: tex1Dfetch()
  - Текстуры на массивах: tex1D() or tex2D() or tex3D()

Texture API

Работа с текстурами (Host)

- Линейная память

Работа с текстурами (Host)

```
texture<float, 2, cudaReadModeElementType> tex;

  ...

cudaChannelFormatDesc channelDesc =
        cudaCreateChannelDesc(32, 0, 0, 0, cudaChannelFormatKindFloat);

cudaArray* cu_arr;

cudaMallocArray(&cu_arr, &channelDesc, width, height );
cudaMemcpyToArray(cu_arr, 0, 0, h_dta, size, cudaMemcpyHostToDevice);

// set texture parameters
  tex.addressMode[0] = cudaAddressModeWrap;
  tex.addressMode[1] = cudaAddressModeWrap;
  tex.filterMode = cudaFilterModeLinear;
  tex.normalized = true;    // access with normalized texture coordinates

// Bind the array to the texture
  cudaBindTextureToArray(tex, cu_arr, channelDesc);
```

Работа с текстурами (Device)

```
__global__ void Kernel( float* g_odata, int width, int height, float theta) {
    // calculate normalized texture coordinates
    unsigned int x = blockIdx.x*blockDim.x + threadIdx.x;
    unsigned int y = blockIdx.y*blockDim.y + threadIdx.y;

    float u = x / (float) width;
    float v = y / (float) height;

    // transform coordinates
    u -= 0.5f;
    v -= 0.5f;

    float tu = u*cosf(theta) - v*sinf(theta) + 0.5f;
    float tv = v*cosf(theta) + u*sinf(theta) + 0.5f;

    // read from texture and write to global memory
    g_odata[y*width + x] = tex2D(tex, tu, tv);
}
```