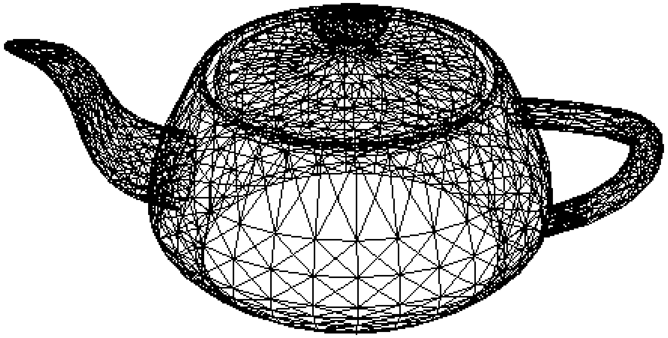
Textures and surfaces

Alexey A. Romanenko

arom@ccfit.nsu.ru

Novosibirsk State University

What is texture?



- Special way to access the data

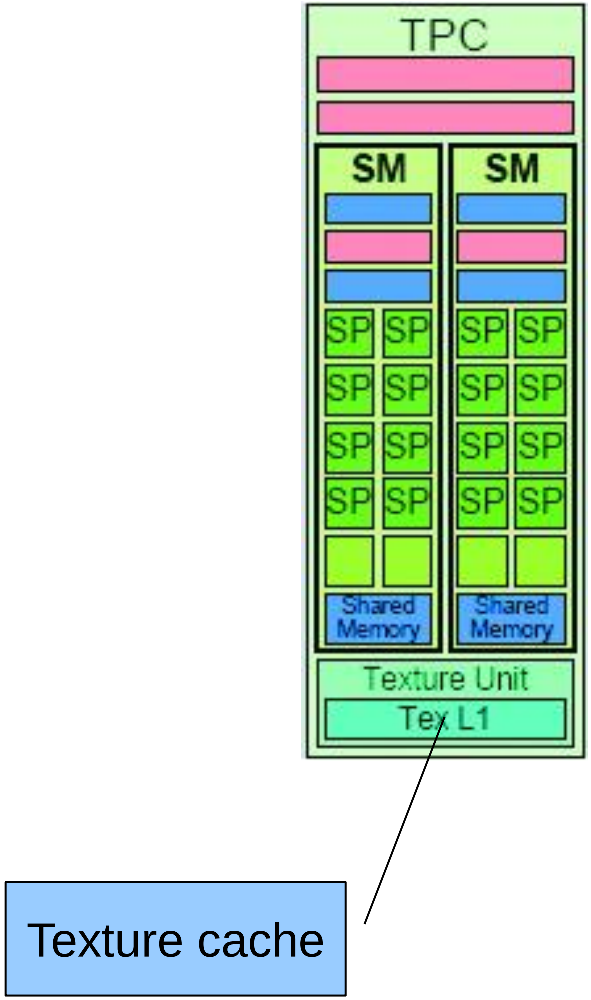
Texture features

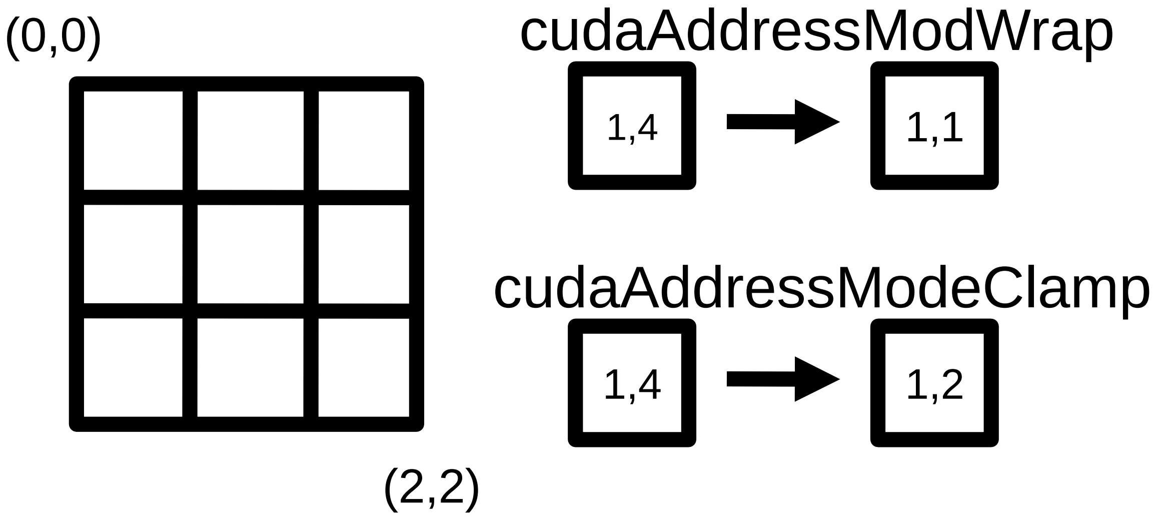
- Texture fetch costs one memory read from device memory only on a cache miss
- Extra pipeline stages:
  - Address translation
  - Filtering
  - Data translation
- But there is a cache
- Recommended if:
  - Data could not be resided in shared memory
  - Random data access pattern (optimized for 2D locality)
  - Threads reuse the same data.

Texture features

- Data are cached
- Filter mode
  - Point/linear
- Address translation
  - Wrap/clamp
- Addressing in 1D, 2D и 3D
- Integer or normalized coordinates

Texture





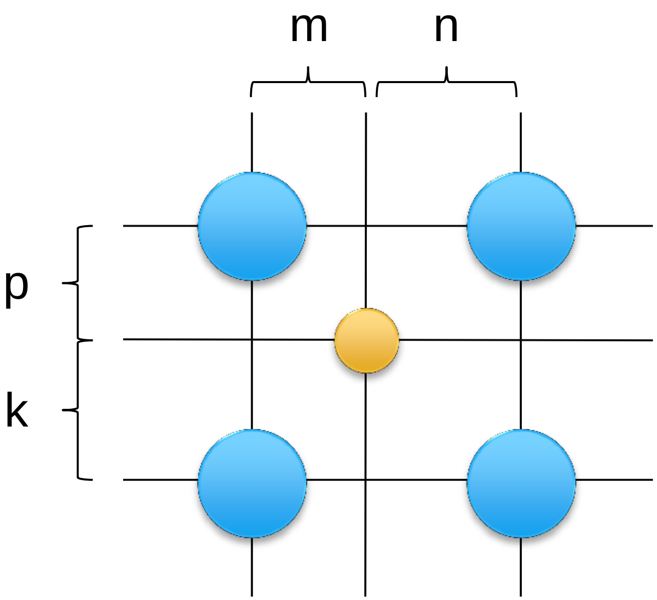
- tex1Dfetch(texRef, x)
- tex1D(texRef, x)
- tex2D(texRef, x, y)
- tex3D(texRef, x, y, z)

Textures

- Normalized coordinates

  [0… n] -> [0… 1]

- Filter mode



$$
\begin{aligned}
U \;=\; & V_{11} * n * k \;+ \\
& V_{12} * m * k \;+ \\
& V_{21} * n * p \;+ \\
& V_{22} * m * p
\end{aligned}
$$

Textures

- Data translation:
  - cudaReadModeNormalizedFloat :
    - Input data are in integer,
    - Output data are in floating ([0, 1] or [-1,1])
  - cudaReadModeElementType
    - Output data are the same as input data

Texture binding

- Binding to linear memory
  - 1D only
  - Integer addressing
  - No filtering or address translation
- Binding to CUDA arrays
  - 1D, 2D or 3D
  - Integer/normalized coordinates
  - Filtering mode
  - Address translation

Working with textures

- Host:
  - Allocate memory (cudaMalloc/cudaMallocArray/…)
  - Declare texture reference
  - Bind texture to memory
  - Unbind texture:
  - Free memory
- Device:
  - Reading data
    - tex1Dfetch()
    - tex1D() or tex2D() or tex3D()

Working with textures (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);
```

Working with textures (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);
}
```

Double precision and textures

- Textures doesn’t support Double data type
- Double could be presented as int[2]
  - texture<int2,1> my_texture;

```
static __inline__ __device__
    double fetch_double(texture<int2, 1> t, int i) {
            int2 v = tex1Dfetch(t,i);
            return __hiloint2double(v.y, v.x);
}
```

Example

```
__global__ void kern(double *o){
        unsigned int x = blockIdx.x*blockDim.x + threadIdx.x;
        if(x<32){
              o[x] = fetch_double(my_texture, x)*2.0;
        }
}
int main(int argc, char *argv[]){
        double hbuf[32];    double *dob;    double *dbuf;
        size_t ii;
        cudaMalloc((void**)&dbuf, sizeof(double)*32);
        cudaMalloc((void**)&dob, sizeof(double)*32);
        cudaBindTexture(&ii, my_texture, dbuf,
            cudaCreateChannelDesc(32,32,0,0, cudaChannelFormatKindSigned));
        for(i = 0 ; i < 32 ; i++)   hbuf[i]=1.0/3.0*i;
        cudaMemcpy(dbuf, hbuf, 32*sizeof(double), cudaMemcpyHostToDevice);
        kern<<<1, 32>>>(dob);
        cudaMemcpy(hbuf, dob, 32*sizeof(double), cudaMemcpyDeviceToHost);
        for(i = 0 ; i < 32 ; i++)   printf("%lf\t", hbuf[i]);
        printf("\n");
        return 0;
}
```

Surface

- Introduced in CUDA 3.2
- One can read/write data from/to surface.
- Declaration
  - surface<void, Dim> surface_ref;
- Binding to CUDA arrays
  - surface <void, 2> surfRef;
  - cudaBindSurfaceToArray(surfRef, cuArray);

Surface. Addressing

- Byte-addressing
- If we have surface/texture with floats
  - Texture - tex1d(texRef1D, x)
  - Surface - surf1Dread(surfRef1D, 4*x)
  - Texture - tex2d(texRef2D, x, y)
  - Surface - surf2Dread(surfRef2D, 4*x, y)

Example

```
// 2D surfaces
surface<void, 2> inputSurfRef;
surface<void, 2> outputSurfRef;

// Simple copy kernel
__global__ void copyKernel(int width, int height) {
  // Calculate surface coordinates
  unsigned int x = blockIdx.x * blockDim.x + threadIdx.x;
  unsigned int y = blockIdx.y * blockDim.y + threadIdx.y;
  if (x < width && y < height) {
    uchar4 data;
    // Read from input surface
    surf2Dread(&data, inputSurfRef, x * 4, y);
    // Write to output surface
    surf2Dwrite(data, outputSurfRef, x * 4, y);
  }
}
```

Example (cont.)

```
int main() {
    cudaChannelFormatDesc channelDesc =
    cudaCreateChannelDesc(8,8,8,8,
                          cudaChannelFormatKindUnsigned);
    cudaArray* cuInputArray; cudaArray* cuOutputArray;
    cudaMallocArray(&cuInputArray, &channelDesc, width,
                       height, cudaArraySurfaceLoadStore);
    cudaMallocArray(&cuOutputArray, &channelDesc, width,
                       height, cudaArraySurfaceLoadStore);
    cudaMemcpyToArray(cuInputArray, 0, 0, h_data, size,
                                    cudaMemcpyHostToDevice);
    cudaBindSurfaceToArray(inputSurfRef, cuInputArray);
    cudaBindSurfaceToArray(outputSurfRef, cuOutputArray);
    // Invoke kernel
    dim3 dimB(16, 16);
    dim3 dimG((width + dimB.x - 1)/dimB.x,
              (height + dimB.y - 1)/ dimB.y);

    copyKernel<<<dimGrid, dimBlock>>>(width, height);
    cudaFreeArray(cuInputArray); cudaFreeArray(cuOutputArray);
}
```

Texture/surface types

- Layered texture/surface
- Cubemap texture/surface
- Cubmap layered texture/surface
- Texture gather