

Sparse GPU Voxelization of Yarn-Level Cloth

Supplementary Material

1. OpenGL and CUDA interaction

```

glGenBuffers(1, &modelMatrixbuffer);
glBindBuffer(GL_ARRAY_BUFFER, modelMatrixbuffer);
glBufferData(GL_ARRAY_BUFFER, m_numberOfProfiles* ←
    sizeof(mat4), NULL, GL_DYNAMIC_DRAW);
// Register VBO with CUDA
glBindBuffer(GL_ARRAY_BUFFER, modelMatrixbuffer);
registerGLBufferObject(modelMatrixbuffer, &←
    m_cuda_vbo_resource);
m_slices = (glm::mat4 *)glMapBuffer(GL_ARRAY_BUFFER, ←
    GL_WRITE_ONLY);
// Compute and fill modelview data in m_slices
...
// Now give a handle to the matrices array for the ←
    CUDA sorting algorithms
glUnmapBuffer(GL_ARRAY_BUFFER);
void* d_matrices = mapGLBufferObject(&←
    m_cuda_vbo_resource);
// Sort with CUDA kernels
    
```

2. Bindless Texture Management

```

// create a 3D texture per block
mHandleVolText = glGetImageHandleARB(m_handle, 0, ←
    GL_TRUE, GL_READ_WRITE, GL_R8);
glMakeImageHandleResidentARB(mHandleVolText, GL_READ_WRITE←
    );
// copy the handles to a data buffer
glBindBuffer(GL_TEXTURE_BUFFER, BufferName); glBufferData←
    (GL_TEXTURE_BUFFER,
mCoarseRes*mCoarseRes*mCoarseRes*sizeof(glm::uvec2), &←
    tex_handles[0], GL_STATIC_DRAW); glBindBuffer(←
    GL_TEXTURE_BUFFER, 0);
glGenTextures(1, &BufferTexture);
// put the data buffer into a texture
glBindTexture(GL_TEXTURE_BUFFER, BufferTexture); ←
    glTexBuffer(GL_TEXTURE_BUFFER, GL_RG32UI, BufferName←
    );
    
```

3. Vertex Shader

```

#version 440 core
#extension GL_ARB_bindless_texture : require

// Input vertex data, different for all executions of ←
    this shader.
layout(location = 0) in vec3 vertexPosition_modelspace; ←
    layout(location = 1) in vec2 vertexUV; layout(←
    location = 2) in mat4
model_matrix;
// Output data ; will be interpolated for each fragment.
out vec3 Position_worldspace; out vec2 UV;

out vec4 MVleft; out vec4 MVup; out vec4 MVforward; out ←
    vec4 MVtrans;

out vec3 EyeDirCam; out vec3 LightDirection_cameraspace;
// Values that stay constant for the whole mesh.
uniform mat4 P; uniform mat4 V; uniform mat4 MVP;

uniform vec3 lightDir;

void main(){

    // Output position of the vertex, in clip space : MVP←
    * position
    gl_Position = P * ( (V * model_matrix) * vec4(←
    vertexPosition_modelspace,1));

    // Position of the vertex, in worldspace : M * ←
    position
    Position_worldspace = (mat4(model_matrix) * vec4(←
    vertexPosition_modelspace,1)).xyz;
    MVleft= model_matrix[0];
    MVup= model_matrix[1];
    MVforward= model_matrix[2];
    MVtrans= model_matrix[3];
    UV = vertexUV;

    // Vector that goes from the vertex to the camera, in←
    camera space.
    // In camera space, the camera is at the origin ←
    (0,0,0).
    vec3 vertexPosition_cameraspace = (V * vec4(←
    Position_worldspace,1)).xyz;
    EyeDirCam = vec3(0,0,0) - vertexPosition_cameraspace←
    ;
    vec3 lightDir_cameraspace = (V * vec4(lightDir,1)).←
    xyz;
    LightDirection_cameraspace = lightDir_cameraspace + ←
    EyeDirCam;
}
    
```

4. Fragment Shader

```

#version 440 core #extension GL_ARB_bindless_texture : ←
require

// Interpolated values from the vertex shaders
in vec3 Position_worldspace; in vec2 UV;

in vec4 MVleft; in vec4 MVup; in vec4 MVforward; in vec4 ←
MVtrans;

in vec3 EyeDirCam; in vec3 LightDirection_cameraspace;
// Output data
out vec4 color;

// Values that stay constant for the whole mesh.
uniform sampler2D DiffuseTextureSampler; uniform ←
sampler2D OrientationTextureSampler;
//uniform sampler2D shadowMap; //for shadow mapping
uniform usamplerBuffer tex_handles; uniform ←
usamplerBuffer phi_handles; uniform usamplerBuffer ←
theta_handles;

uniform int coarseRes; uniform int fineRes;

void main(){

float PI = 3.14159265358979323846264f;
vec4 MaterialDiffuseColor = textureLod(←
DiffuseTextureSampler, UV,0).rgba;

//fetch //((z * m_res.y) + y) * m_res.x + x
int x3D= int( Position_worldspace.x* float(coarseRes)←
);
int y3D= int( Position_worldspace.y* float(coarseRes←
));
int z3D= int( Position_worldspace.z* float(coarseRes)←
);

//Grid: coarse level index
int index= ((z3D * coarseRes) + y3D) * coarseRes + ←
x3D;

uvec4 aux = texelFetch(tex_handles, index);
uvec2 vol_handle = aux.rg;
layout(r8) image3D volDensity= layout(r8) image3D(←
vol_handle);

aux = texelFetch(theta_handles, index);
vol_handle = aux.rg;
layout(r8) image3D volTheta= layout(r8) image3D(←
theta_handles);

aux = texelFetch(phi_handles, index);
vol_handle = aux.rg;
layout(r8) image3D volPhi= layout(r8) image3D(←
phi_handles);

//get intra-block fine resolution coords
int x3Dfine= int(Position_worldspace.x*(coarseRes*←
fineRes)) - x3D*fineRes;
int y3Dfine= int(Position_worldspace.y*(coarseRes*←
fineRes)) - y3D*fineRes;
int z3Dfine= int(Position_worldspace.z*(coarseRes*←
fineRes)) - z3D*fineRes;

//To load data from the image: imageLoad(volDensity, ←
ivec3(x3Dfine, y3Dfine, z3Dfine));
imageStore(volDensity, ivec3(x3Dfine, y3Dfine, z3Dfine)←
, vec4(1.0f));

//Orientation computation & storage
vec3 dOrient = texture2D(OrientationTextureSampler, ←
UV).xyz;
vec4 gOrient= vec4(normalize((dOrient.x*MVleft + ←
dOrient.y*MVup + dOrient.z*MVforward).xyz), 1.0f←
);
float theta, phi;
theta = acos(gOrient.z);
theta = theta/PI; //((theta+PI)/(2*PI); //for theta ←
in -pi, pi

if (gOrient.x == 0.0f){
phi = 0.0f;

```

```

} else{
phi = atan(gOrient.y, gOrient.x);
phi= (phi+PI)/(2*PI);
}

float tmpTheta = imageLoad(volTheta, ivec3(x3Dfine, ←
y3Dfine, z3Dfine));
float tmpPhi = imageLoad(volPhi, ivec3(x3Dfine, ←
y3Dfine, z3Dfine));
//A special value (e.g.:0) is used as non-initialized←
tag
// → to avoid averaging the first hit (←
conditional not listed)
imageStore(volTheta, ivec3(x3Dfine, y3Dfine, z3Dfine), ←
vec4((theta+tmpTheta)/2.0));
imageStore(volPhi, ivec3(x3Dfine, y3Dfine, z3Dfine), ←
vec4((phi+tmpPhi)/2.0));

//kajiya-Kay shading
vec3 L =normalize(-LightDirection_cameraspace);
vec3 N = normalize(gOrient.xyz);
vec3 E = normalize(EyeDirCam);
vec3 H= vec3(MaterialDiffuseColor.xyz);
float dotTL=dot(N,L);
float dotTV=dot(N,E);
float sinTL=sqrt(1.0-dotTL*dotTL);
float sinTV=sqrt(1.0-dotTV*dotTV);
vec3 shadowColor = vec3(0.0f,0.1f,0.8f);
float dotL2N=sqrt(1.0-dot(L,N)*dot(L,N));
H=1.2f* (dotL2N*H + (1-dotL2N)*shadowColor)/2.0f; //←
shading
H+=0.3f*pow( max((dotTL*dotTV +sinTL*sinTV),0.0), ←
10.0f ); //specular

color=vec4(H.xyz,MaterialDiffuseColor.a);
}

```