Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Чувашский государственный университет имени И.Н. Ульянова»

ФАКУЛЬТЕТ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Кафедравычислительной техники

АГС.

Лабораторная работа №5.

«Использование UBO для передачи

uniform-переменных в шейдер.»

Выполнил: студент гр. ИВТ-41-15

Лазарев Дмитрий

Проверил:

Ст. преп. Галибин С.В.

Чебоксары – 2018

В лабораторной работе №5 необходимо:

1. Реализовать классы CRenderManager вышеописанной структуры.

2. Класс должен быть реализован на основе шаблона singleton.

3. Класс должен реализовывать оптимизацию с использование UBO-буферов.

4. Класс должен подсчитывать количество обновлений буферов UBO.

5. В заголовке окна необходимо вывести текущее количество FPS и количество обновлений UBO-буферов.

6. Реализовать вывод нескольких (не менее 5) объектов, часть из которых является одинаковыми.

#include "CRenderManager.h"

int CRenderManager::UBOUpdateCount = 0;

void CRenderManager::init(void)

{

Shader.LoadVertexShader((char\*)"SHADER\\DirectLight.vsh");

Shader.LoadFragmentShader((char\*)"SHADER\\DirectLight.fsh");

Shader.Link();

createPerSceneBlock();

}

void CRenderManager::start(void)

{

glClearColor(0.0, 0.0, 0.0, 1.0);

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glEnable(GL\_DEPTH\_TEST);

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);

glEnable(GL\_CULL\_FACE);

glCullFace(GL\_BACK);

GraphicObjects.clear();

}

void CRenderManager::setCamera(CCamera & Camera)

{

if (this->Camera != Camera)

{

this->Camera = Camera;

updatePerSceneBlock(PerSceneUBOIndex);

for (auto it = RenderManagerObjectStates.begin();

it != RenderManagerObjectStates.end(); it++)

{

it->second.UBOIndexUpdate = true;

};

};

}

void CRenderManager::setLight(CLight & Light)

{

if (this->Light != Light)

{

this->Light = Light;

updatePerSceneBlock(PerSceneUBOIndex);

for (auto it = RenderManagerObjectStates.begin();

it != RenderManagerObjectStates.end(); it++)

{

it->second.UBOIndexUpdate = true;

};

};

}

void CRenderManager::addToRenderQueue(CGraphicObject & GraphicObject)

{

GraphicObjects.push\_back(GraphicObject);

auto it = RenderManagerObjectStates.find(GraphicObject.getID());

if (it == RenderManagerObjectStates.end())

{

GLuint index = createPerObjectBlock(GraphicObject);

SRenderManagerObjectState state =

{

GraphicObject,

false,

index

};

RenderManagerObjectStates.insert(std::pair<int, SRenderManagerObjectState>(GraphicObject.getID(),state));

}

}

void CRenderManager::finish(void)

{

Shader.Activate();

glBindBuffer(GL\_UNIFORM\_BUFFER, PerSceneUBOIndex);

glBindBufferBase(GL\_UNIFORM\_BUFFER, 0, PerSceneUBOIndex);

// получаем матрицу проекции

for (int i = 0; i < GraphicObjects.size(); i++)

{

if (RenderManagerObjectStates[GraphicObjects[i].getID()].UBOIndexUpdate)

{

updatePerObjectBlock(RenderManagerObjectStates[GraphicObjects[i].getID()].UBOIndex, GraphicObjects[i]);

RenderManagerObjectStates[GraphicObjects[i].getID()].UBOIndexUpdate = false;

}

glBindBufferBase(GL\_UNIFORM\_BUFFER, 1, RenderManagerObjectStates[GraphicObjects[i].getID()].UBOIndex);

CMesh\* mesh = CResourceManager::Instance().GetMesh(GraphicObjects[i].getMesh());

if (mesh != nullptr) mesh->Render();

}

CShader::Deactivate();

}

GLuint CRenderManager::createPerSceneBlock()

{

glGenBuffers(1, &PerSceneUBOIndex);

glBindBuffer(GL\_UNIFORM\_BUFFER, PerSceneUBOIndex);

glBufferData(GL\_UNIFORM\_BUFFER, sizeof(PerSceneBlock), nullptr, GL\_STATIC\_DRAW);

glBindBuffer(GL\_UNIFORM\_BUFFER, 0);

UBOUpdateCount++;

return PerSceneUBOIndex;

}

GLuint CRenderManager::createPerObjectBlock(CGraphicObject& GraphicObject)

{

PerObjectBlock temp =

{

Camera.GetViewMatrix()\*GraphicObject.getModelMatrix(),

GraphicObject.getMaterial().getAmbient(),

GraphicObject.getMaterial().getDiffuse(),

GraphicObject.getMaterial().getSpecular()

};

GLuint tempIndex;

glGenBuffers(1, &tempIndex);

glBindBuffer(GL\_UNIFORM\_BUFFER, tempIndex);

glBufferData(GL\_UNIFORM\_BUFFER, sizeof(PerObjectBlock), &temp, GL\_STATIC\_DRAW);

glBindBuffer(GL\_UNIFORM\_BUFFER, 0);

UBOUpdateCount++;

return tempIndex;

}

void CRenderManager::updatePerSceneBlock(int UBOindex)

{

PerSceneBlock temp =

{

Camera.GetProjectionMatrix(),

Light.getAmbient(),

Light.getDiffuse(),

Light.getSpecular(),

Camera.GetViewMatrix()\*Light.getPosition()

};

glBindBuffer(GL\_UNIFORM\_BUFFER, UBOindex);

glBufferData(GL\_UNIFORM\_BUFFER, sizeof(PerSceneBlock), &temp, GL\_STATIC\_DRAW);

glBindBuffer(GL\_UNIFORM\_BUFFER, 0);

UBOUpdateCount++;

return;

}

void CRenderManager::updatePerObjectBlock(int UBOindex, CGraphicObject& GraphicObject)

{

PerObjectBlock temp =

{

Camera.GetViewMatrix()\*GraphicObject.getModelMatrix(),

GraphicObject.getMaterial().getAmbient(),

GraphicObject.getMaterial().getDiffuse(),

GraphicObject.getMaterial().getSpecular()

};

glBindBuffer(GL\_UNIFORM\_BUFFER, UBOindex);

glBufferData(GL\_UNIFORM\_BUFFER, sizeof(PerObjectBlock), &temp, GL\_STATIC\_DRAW);

glBindBuffer(GL\_UNIFORM\_BUFFER, 0);

UBOUpdateCount++;

return;

}

|  |  |
| --- | --- |
| Без оптимизации | С оптимизацией |
| 700 | 750 |