#define M\_PI 3.14159F

#define ToRadian(x) ((x) \* M\_PI / 180.0f)

#define ToDegree(x) ((x) \* 180.0f / M\_PI)

#include "pipeline3.h"

#include "mat4,vec3.h"

/\*vec3 vec3::Cross(const vec3& v) const

{

const float \_x = y \* v.z - z \* v.y;

const float \_y = z \* v.x - x \* v.z;

const float \_z = x \* v.y - y \* v.x;

return vec3(\_x, \_y, \_z);

}\*/

/\*vec3& vec3::Normalize()

{

const float Length = sqrtf(x \* x + y \* y + z \* z);

x /= Length;

y /= Length;

z /= Length;

return \*this;

}\*/

//void Pipeline::InitCameraTransform(mat4& m, const vec3& Target, const vec3& Up)

//{

// vec3 N = Target;

// //N.Normalize();

// vec3 N = normalize(N);

// vec3 U = Up;

// //U.Normalize();

// vec3 U = normalize(U);

// //U = U.Cross(Target);

// U = cross(Target, U);

// //vec3 V = N.Cross(U);

// vec3 V = cross(U, N);

//

// m[0][0] = U.x; m[0][1] = U.y; m[0][2] = U.z; m[0][3] = 0.0f;

// m[1][0] = V.x; m[1][1] = V.y; m[1][2] = V.z; m[1][3] = 0.0f;

// m[2][0] = N.x; m[2][1] = N.y; m[2][2] = N.z; m[2][3] = 0.0f;

// m[3][0] = 0.0f; m[3][1] = 0.0f; m[3][2] = 0.0f; m[3][3] = 1.0f;

//}

void Pipeline::InitScaleTransform(mat4& m) const

{

m[0][0] = m\_scale.x; m[0][1] = 0.0f; m[0][2] = 0.0f; m[0][3] = 0.0f;

m[1][0] = 0.0f; m[1][1] = m\_scale.y; m[1][2] = 0.0f; m[1][3] = 0.0f;

m[2][0] = 0.0f; m[2][1] = 0.0f; m[2][2] = m\_scale.z; m[2][3] = 0.0f;

m[3][0] = 0.0f; m[3][1] = 0.0f; m[3][2] = 0.0f; m[3][3] = 1.0f;

}

void Pipeline::InitRotateTransform(mat4& m) const

{

mat4 rx, ry, rz;

const float x = ToRadian(m\_rotateInfo.x);

const float y = ToRadian(m\_rotateInfo.y);

const float z = ToRadian(m\_rotateInfo.z);

rx[0][0] = 1.0f; rx[0][1] = 0.0f; rx[0][2] = 0.0f; rx[0][3] = 0.0f;

rx[1][0] = 0.0f; rx[1][1] = cosf(x); rx[1][2] = -sinf(x); rx[1][3] = 0.0f;

rx[2][0] = 0.0f; rx[2][1] = sinf(x); rx[2][2] = cosf(x); rx[2][3] = 0.0f;

rx[3][0] = 0.0f; rx[3][1] = 0.0f; rx[3][2] = 0.0f; rx[3][3] = 1.0f;

ry[0][0] = cosf(y); ry[0][1] = 0.0f; ry[0][2] = -sinf(y); ry[0][3] = 0.0f;

ry[1][0] = 0.0f; ry[1][1] = 1.0f; ry[1][2] = 0.0f; ry[1][3] = 0.0f;

ry[2][0] = sinf(y); ry[2][1] = 0.0f; ry[2][2] = cosf(y); ry[2][3] = 0.0f;

ry[3][0] = 0.0f; ry[3][1] = 0.0f; ry[3][2] = 0.0f; ry[3][3] = 1.0f;

rz[0][0] = cosf(z); rz[0][1] = -sinf(z); rz[0][2] = 0.0f; rz[0][3] = 0.0f;

rz[1][0] = sinf(z); rz[1][1] = cosf(z); rz[1][2] = 0.0f; rz[1][3] = 0.0f;

rz[2][0] = 0.0f; rz[2][1] = 0.0f; rz[2][2] = 1.0f; rz[2][3] = 0.0f;

rz[3][0] = 0.0f; rz[3][1] = 0.0f; rz[3][2] = 0.0f; rz[3][3] = 1.0f;

m = rz \* ry \* rx; //объединение преобразований

}

void Pipeline::InitTranslationTransform(mat4& m) const

{

m[0][0] = 1.0f; m[0][1] = 0.0f; m[0][2] = 0.0f; m[0][3] = m\_worldPos.x;

m[1][0] = 0.0f; m[1][1] = 1.0f; m[1][2] = 0.0f; m[1][3] = m\_worldPos.y;

m[2][0] = 0.0f; m[2][1] = 0.0f; m[2][2] = 1.0f; m[2][3] = m\_worldPos.z;

m[3][0] = 0.0f; m[3][1] = 0.0f; m[3][2] = 0.0f; m[3][3] = 1.0f;

}

void Pipeline::InitPerspectiveProj(mat4& m) const

{

const float ar = m\_persProj.Width / m\_persProj.Height;

const float zNear = m\_persProj.zNear;

const float zFar = m\_persProj.zFar;

const float zRange = zNear - zFar;

const float tanHalfFOV = tanf(ToRadian(m\_persProj.FOV / 2.0f));

m[0][0] = 1.0f / (tanHalfFOV \* ar); m[0][1] = 0.0f; m[0][2] = 0.0f; m[0][3] = 0.0;

m[1][0] = 0.0f; m[1][1] = 1.0f / tanHalfFOV; m[1][2] = 0.0f; m[1][3] = 0.0;

m[2][0] = 0.0f; m[2][1] = 0.0f; m[2][2] = (-zNear - zFar) / zRange; m[2][3] = 2.0f \* zFar\*zNear / zRange;

m[3][0] = 0.0f; m[3][1] = 0.0f; m[3][2] = 1.0f; m[3][3] = 0.0;

}

/\*const mat4\* Pipeline::GetTrans()

{

mat4 ScaleTrans, RotateTrans, TranslationTrans, PersProjTrans;

InitScaleTransform(ScaleTrans);

InitRotateTransform(RotateTrans);

InitTranslationTransform(TranslationTrans);

InitPerspectiveProj(PersProjTrans);

m\_transformation = RotateTrans \* TranslationTrans \* PersProjTrans \* ScaleTrans;//тпрс было

return &m\_transformation;

}\*/

const mat4\* Pipeline::GetTrans()

{

mat4 ScaleTrans, RotateTrans, TranslationTrans, CameraTranslationTrans, CameraRotateTrans, PersProjTrans;

/\*ScaleTrans.InitScaleTransform(m\_scale.x, m\_scale.y, m\_scale.z);

RotateTrans.InitRotateTransform(m\_rotateInfo.x, m\_rotateInfo.y, m\_rotateInfo.z);

TranslationTrans.InitTranslationTransform(m\_worldPos.x, m\_worldPos.y, m\_worldPos.z);

CameraTranslationTrans.InitTranslationTransform(-m\_camera.Pos.x, -m\_camera.Pos.y, -m\_camera.Pos.z);

CameraRotateTrans.InitCameraTransform(m\_camera.Target, m\_camera.Up);

\*/

InitScaleTransform(ScaleTrans);

InitRotateTransform(RotateTrans);

InitTranslationTransform(TranslationTrans);

InitTranslationTransform(CameraTranslationTrans);

InitCameraTransform(CameraRotateTrans, m\_camera.Target, m\_camera.Up);

/\*PersProjTrans.InitPersProjTransform(m\_persProj.FOV, m\_persProj.Width,

m\_persProj.Height, m\_persProj.zNear, m\_persProj.zFar);

InitPersProjTransform(PersProjTrans);\*/

InitPerspectiveProj(PersProjTrans);

m\_transformation = PersProjTrans \* CameraRotateTrans \* CameraTranslationTrans \* TranslationTrans \* RotateTrans \* ScaleTrans;

return &m\_transformation;

}

//void InitScaleTransform(float ScaleX, float ScaleY, float ScaleZ);

//void InitRotateTransform(float RotateX, float RotateY, float RotateZ);

//void InitTranslationTransform(float x, float y, float z);

//void InitCameraTransform(const vec3& Target, const vec3& Up);

//void InitPersProjTransform(float FOV, float Width, float Height, float zNear, float zFar);