Lab #4 – myTransform

Computer Graphics

컴퓨터 그래픽스

남병규 교수님

201002506 진재연

2014. 10. 22

# Lab의 목적

이번 과제의 목적은 저번 과제에서 다뤘던 OpenGL의 행렬 스택을 이해하고 각 변환 행렬의 모양을 익혀서 직접 그래픽 파이프라인에 사용되는 행렬 스택을 구현해볼 수 있는 것입니다.

# 소스 코드

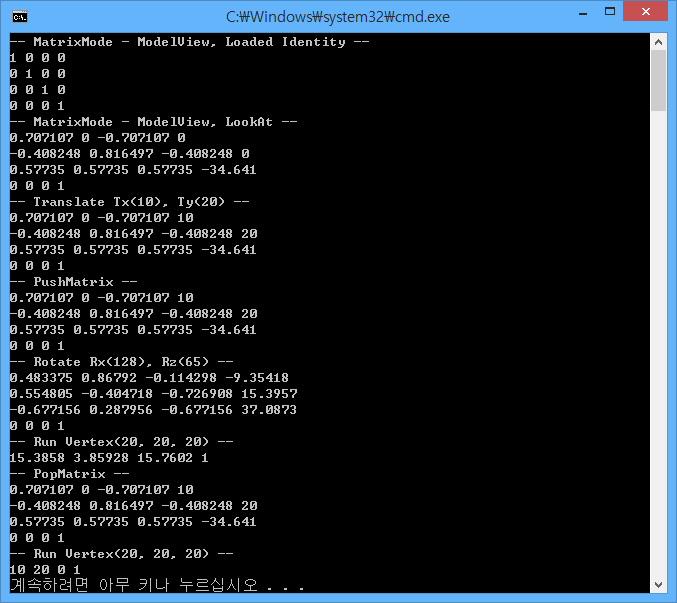
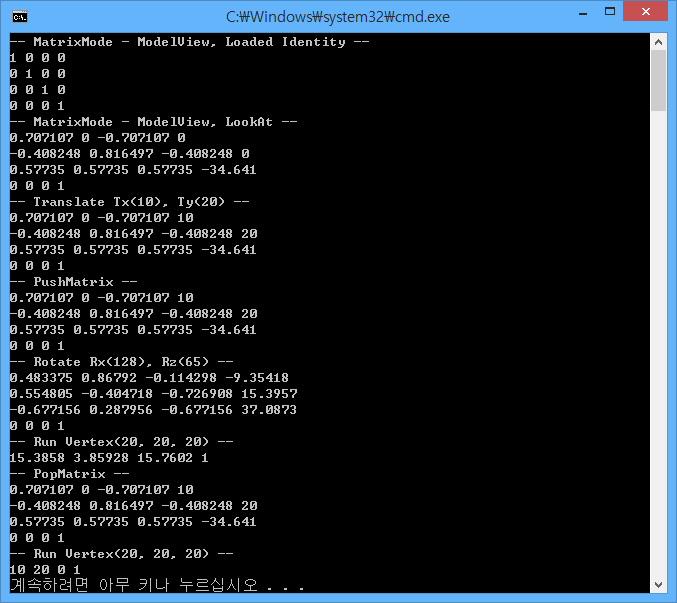
|  |
| --- |
| ***MyMatrix.h*** |
| #ifndef \_\_MatrixLib\_\_MyMatrix\_\_  #define \_\_MatrixLib\_\_MyMatrix\_\_  namespace daram {  struct Vector2;  struct Vector3;  struct Vector4;  struct Matrix4x4;  struct Vector2  {  public:  Vector2 ();  Vector2 ( float x, float y );  public:  Vector2 operator+ ( Vector2 & v );  Vector2 operator- ( Vector2 & v );  Vector2 operator- ();  Vector2 operator\* ( Vector2 & v );  Vector2 operator/ ( Vector2 & v );  bool operator== ( Vector2 & v );  public:  float length ();  Vector2 normalize ();  public:  float dot ( Vector2 & v );  Vector2 cross ( Vector2 & v );  public:  float x, y;  };  struct Vector3  {  public:  Vector3 ();  Vector3 ( float x, float y, float z );  Vector3 ( Vector2 & vector, float z );  public:  Vector3 operator+ ( Vector3 & v );  Vector3 operator- ( Vector3 & v );  Vector3 operator- ( );  Vector3 operator\* ( Vector3 & v );  Vector3 operator/ ( Vector3 & v );  bool operator== ( Vector3 & v );  public:  float length ();  Vector3 normalize ();  public:  float dot ( Vector3 & v );  Vector3 cross ( Vector3 & v );  public:  float x, y, z;  };  struct Vector4  {  public:  Vector4 ();  Vector4 ( float x, float y, float z, float w );  Vector4 ( Vector3 & vector, float w );  public:  Vector4 operator+ ( Vector4 & v );  Vector4 operator- ( Vector4 & v );  Vector4 operator- ( );  Vector4 operator\* ( Vector4 & v );  Vector4 operator\* ( Matrix4x4 & matrix );  Vector4 operator/ ( Vector4 & v );  bool operator== ( Vector4 & v );  public:  float length ();  Vector4 normalize ();  public:  float dot ( Vector4 & v );  Vector4 cross ( Vector4 & v1, Vector4 & v2 );  public:  float x, y, z, w;  };  struct Matrix4x4 {  public:  Matrix4x4 ();  Matrix4x4 ( float m11, float m12, float m13, float m14,  float m21, float m22, float m23, float m24,  float m31, float m32, float m33, float m34,  float m41, float m42, float m43, float m44 );  Matrix4x4 ( Vector4 & column1, Vector4 & column2, Vector4 & column3, Vector4 & column4 );  public:  Matrix4x4 operator+ ( Matrix4x4 & v );  Matrix4x4 operator+ ( float v );  Matrix4x4 operator- ( Matrix4x4 & v );  Matrix4x4 operator- ( float v );  Matrix4x4 operator\* ( Matrix4x4 & v );  Matrix4x4 operator\* ( float v );  Matrix4x4 operator/ ( Matrix4x4 & v );  Matrix4x4 operator/ ( float v );  bool operator== ( Matrix4x4 & v );  public:  Matrix4x4 transpose ();  Matrix4x4 invert ();  float determinant ();  public:  static Matrix4x4 identity ();  static Matrix4x4 translate ( Vector3 & translate );  static Matrix4x4 scale ( Vector3 & scale );  static Matrix4x4 rotateX ( float r );  static Matrix4x4 rotateY ( float r );  static Matrix4x4 rotateZ ( float r );  static Matrix4x4 orthographicOffCenterLH ( float l, float r, float b, float t, float zn, float zf );  static Matrix4x4 orthographicOffCenterRH ( float l, float r, float b, float t, float zn, float zf );  static Matrix4x4 perspectiveFieldOfViewLH ( float fov, float aspect, float zn, float zf );  static Matrix4x4 perspectiveFieldOfViewRH ( float fov, float aspect, float zn, float zf );  static Matrix4x4 lookAtLH ( Vector3 & eye, Vector3 & at, Vector3 & up );  static Matrix4x4 lookAtRH ( Vector3 & eye, Vector3 & at, Vector3 & up );  public:  float m11, m12, m13, m14, m21, m22, m23, m24, m31, m32, m33, m34, m41, m42, m43, m44;  };  enum MatrixMode {  MatrixMode\_ModelView,  MatrixMode\_Projection,  };  void mySetMatrixMode(MatrixMode matrixMode);  void myLoadIdentity();  void myPushMatrix();  void myPopMatrix();  void myMatrixMul(Matrix4x4 & matrix);  void myRunMatrix(Vector3 & vertex);  void myTranslateX(float tx);  void myTranslateY(float ty);  void myTranslateZ(float tz);  void myRotateX(float rx);  void myRotateY(float ry);  void myRotateZ(float rz);  void myScaleX(float sx);  void myScaleY(float sy);  void myScaleZ(float sz);  void myLookAt(Vector3 & eye, Vector3 & at, Vector3 & up);  Matrix4x4 getTopMatrix();  Vector4 getRunnedVertex();  void showMatrix4x4(Matrix4x4 & matrix);  void showVector4(Vector4 & vector);  }  #endif |
| ***MyMatrix.cpp*** |
| #include "MyMatrix.h"  #include <cmath>  using namespace daram;  Vector2::Vector2 () { }  Vector2::Vector2 ( float x, float y ) : x ( x ), y ( y ) { }  Vector2 Vector2::operator+ ( Vector2 & v ) { return Vector2 ( x + v.x, y + v.y ); }  Vector2 Vector2::operator- ( Vector2 & v ) { return Vector2 ( x - v.x, y - v.y ); }  Vector2 Vector2::operator- ( ) { return Vector2 ( -x, -y ); }  Vector2 Vector2::operator\* ( Vector2 & v ) { return Vector2 ( x \* v.x, y \* v.y ); }  Vector2 Vector2::operator/ ( Vector2 & v ) { return Vector2 ( x / v.x, y / v.y ); }  bool Vector2::operator== ( Vector2 & v ) { return x == v.x && y == v.y; }  float Vector2::length () { return sqrtf ( x \* x + y \* y ); }  Vector2 Vector2::normalize () { float len = length (); return Vector2 ( x / len, y / len ); }  float Vector2::dot ( Vector2 & v ) { return x \* v.x + y \* v.y; }  Vector2 Vector2::cross ( Vector2 & v ) { return Vector2 ( x \* v.y, y \* v.x ); }  Vector3::Vector3 () { }  Vector3::Vector3 ( float x, float y, float z ) : x ( x ), y ( y ), z ( z ) { }  Vector3::Vector3 ( Vector2 & vector, float z ) : x ( vector.x ), y ( vector.y ), z ( z ) { }  Vector3 Vector3::operator+ ( Vector3 & v ) { return Vector3 ( x + v.x, y + v.y, z + v.z ); }  Vector3 Vector3::operator- ( Vector3 & v ) { return Vector3 ( x - v.x, y - v.y, z - v.z ); }  Vector3 Vector3::operator- ( ) { return Vector3 ( -x, -y, -z ); }  Vector3 Vector3::operator\* ( Vector3 & v ) { return Vector3 ( x \* v.x, y \* v.y, z \* v.z ); }  Vector3 Vector3::operator/ ( Vector3 & v ) { return Vector3 ( x / v.x, y / v.y, z / v.z ); }  bool Vector3::operator== ( Vector3 & v ) { return x == v.x && y == v.y && z == v.z; }  float Vector3::length () { return sqrtf ( x \* x + y \* y + z \* z ); }  Vector3 Vector3::normalize () { float len = length (); return Vector3 ( x / len, y / len, z / len ); }  float Vector3::dot ( Vector3 & v ) { return x \* v.x + y \* v.y + z \* v.z; }  Vector3 Vector3::cross ( Vector3 & v ) { return Vector3 ( y \* v.z - z \* v.y, z \* v.x - x \* v.z, x \* v.y - y \* v.x ); }  Vector4::Vector4 () { }  Vector4::Vector4 ( float x, float y, float z, float w ) : x ( x ), y ( y ), z ( z ), w ( w ) { }  Vector4::Vector4 ( Vector3 & vector, float w ) : x ( vector.x ), y ( vector.y ), z ( vector.z ), w ( w ) { }  Vector4::Vector4 ( Quaternion & quaternion ) : x ( quaternion.x ), y ( quaternion.y ), z ( quaternion.z ), w ( quaternion.w ) { }  Vector4 Vector4::operator+ ( Vector4 & v ) { return Vector4 ( x + v.x, y + v.y, z + v.z, w + v.w ); }  Vector4 Vector4::operator- ( Vector4 & v ) { return Vector4 ( x - v.x, y - v.y, z - v.z, w - v.w ); }  Vector4 Vector4::operator- ( ) { return Vector4 ( -x, -y, -z, -w ); }  Vector4 Vector4::operator\* ( Vector4 & v ) { return Vector4 ( x \* v.x, y \* v.y, z \* v.z, w \* v.w ); }  Vector4 Vector4::operator\* ( Matrix4x4 & matrix )  {  return Vector4 ( ( x \* matrix.m11 ) + ( y \* matrix.m21 ) + ( z \* matrix.m31 ) + ( w \* matrix.m41 ),  ( x \* matrix.m12 ) + ( y \* matrix.m22 ) + ( z \* matrix.m32 ) + ( w \* matrix.m42 ),  ( x \* matrix.m13 ) + ( y \* matrix.m23 ) + ( z \* matrix.m33 ) + ( w \* matrix.m43 ),  ( x \* matrix.m14 ) + ( y \* matrix.m24 ) + ( z \* matrix.m34 ) + ( w \* matrix.m44 ) );  }  Vector4 Vector4::operator/ ( Vector4 & v ) { return Vector4 ( x / v.x, y / v.y, z / v.z, w / v.w ); }  bool Vector4::operator== ( Vector4 & v ) { return x == v.x && y == v.y && z == v.z && w == v.w; }  float Vector4::length () { return sqrtf ( x \* x + y \* y + z \* z + w \* w ); }  Vector4 Vector4::normalize () { float len = length (); return Vector4 ( x / len, y / len, z / len, w / len ); }  float Vector4::dot ( Vector4 & v ) { return x \* v.x + y \* v.y + z \* v.z + w \* v.w; }  Vector4 Vector4::cross ( Vector4 & v2, Vector4 & v3 ) {  return Vector4 (  w \* v3.y - v2.w \* v3.z + w \* v3.w,  -w \* v3.x + x \* y \* v3.z - v2.w \* v3.w,  v2.w \* v3.x - x \* y \* v3.y + w \* v3.w,  -w \* v3.x + v2.w \* v3.y - w - v3.z );  }  Matrix4x4::Matrix4x4 () : m11 ( 0 ), m12 ( 0 ), m13 ( 0 ), m14 ( 0 ),  m21 ( 0 ), m22 ( 0 ), m23 ( 0 ), m24 ( 0 ),  m31 ( 0 ), m32 ( 0 ), m33 ( 0 ), m34 ( 0 ),  m41 ( 0 ), m42 ( 0 ), m43 ( 0 ), m44 ( 0 )  { }  Matrix4x4::Matrix4x4 ( float m11, float m12, float m13, float m14,  float m21, float m22, float m23, float m24,  float m31, float m32, float m33, float m34,  float m41, float m42, float m43, float m44 )  : m11 ( m11 ), m12 ( m12 ), m13 ( m13 ), m14 ( m14 ),  m21 ( m21 ), m22 ( m22 ), m23 ( m23 ), m24 ( m24 ),  m31 ( m31 ), m32 ( m32 ), m33 ( m33 ), m34 ( m34 ),  m41 ( m41 ), m42 ( m42 ), m43 ( m43 ), m44 ( m44 ) { }  Matrix4x4::Matrix4x4 ( Vector4 & column1, Vector4 & column2, Vector4 & column3, Vector4 & column4 )  : m11 ( column1.x ), m12 ( column1.y ), m13 ( column1.z ), m14 ( column1.w ),  m21 ( column2.x ), m22 ( column2.y ), m23 ( column2.z ), m24 ( column2.w ),  m31 ( column3.x ), m32 ( column3.y ), m33 ( column3.z ), m34 ( column3.w ),  m41 ( column4.x ), m42 ( column4.y ), m43 ( column4.z ), m44 ( column4.w ) { }  Matrix4x4 Matrix4x4::operator+ ( Matrix4x4 & v ) {  return Matrix4x4 ( m11 + v.m11, m12 + v.m12, m13 + v.m13, m14 + v.m14, m21 + v.m21, m22 + v.m22, m23 + v.m23, m24 + v.m24, m31 + v.m31, m32 + v.m32, m33 + v.m33, m34 + v.m34, m41 + v.m41, m42 + v.m42, m43 + v.m43, m44 + v.m44 );  }  Matrix4x4 Matrix4x4::operator+ ( float v ) {  return Matrix4x4 ( m11 + v, m12 + v, m13 + v, m14 + v, m21 + v, m22 + v, m23 + v, m24 + v, m31 + v, m32 + v, m33 + v, m34 + v, m41 + v, m42 + v, m43 + v, m44 + v );  }  Matrix4x4 Matrix4x4::operator- ( Matrix4x4 & v ) {  return Matrix4x4 ( m11 - v.m11, m12 - v.m12, m13 - v.m13, m14 - v.m14, m21 - v.m21, m22 - v.m22, m23 - v.m23, m24 - v.m24, m31 - v.m31, m32 - v.m32, m33 - v.m33, m34 - v.m34, m41 - v.m41, m42 - v.m42, m43 - v.m43, m44 - v.m44 );  }  Matrix4x4 Matrix4x4::operator- ( float v ) {  return Matrix4x4 ( m11 - v, m12 - v, m13 - v, m14 - v, m21 - v, m22 - v, m23 - v, m24 - v, m31 - v, m32 - v, m33 - v, m34 - v, m41 - v, m42 - v, m43 - v, m44 - v );  }  Matrix4x4 Matrix4x4::operator\* ( Matrix4x4 & v ) {  float \_m11 = ( ( ( m11 \* v.m11 ) + ( m12 \* v.m21 ) ) + ( m13 \* v.m31 ) ) + ( m14 \* v.m41 );  float \_m12 = ( ( ( m11 \* v.m12 ) + ( m12 \* v.m22 ) ) + ( m13 \* v.m32 ) ) + ( m14 \* v.m42 );  float \_m13 = ( ( ( m11 \* v.m13 ) + ( m12 \* v.m23 ) ) + ( m13 \* v.m33 ) ) + ( m14 \* v.m43 );  float \_m14 = ( ( ( m11 \* v.m14 ) + ( m12 \* v.m24 ) ) + ( m13 \* v.m34 ) ) + ( m14 \* v.m44 );  float \_m21 = ( ( ( m21 \* v.m11 ) + ( m22 \* v.m21 ) ) + ( m23 \* v.m31 ) ) + ( m24 \* v.m41 );  float \_m22 = ( ( ( m21 \* v.m12 ) + ( m22 \* v.m22 ) ) + ( m23 \* v.m32 ) ) + ( m24 \* v.m42 );  float \_m23 = ( ( ( m21 \* v.m13 ) + ( m22 \* v.m23 ) ) + ( m23 \* v.m33 ) ) + ( m24 \* v.m43 );  float \_m24 = ( ( ( m21 \* v.m14 ) + ( m22 \* v.m24 ) ) + ( m23 \* v.m34 ) ) + ( m24 \* v.m44 );  float \_m31 = ( ( ( m31 \* v.m11 ) + ( m32 \* v.m21 ) ) + ( m33 \* v.m31 ) ) + ( m34 \* v.m41 );  float \_m32 = ( ( ( m31 \* v.m12 ) + ( m32 \* v.m22 ) ) + ( m33 \* v.m32 ) ) + ( m34 \* v.m42 );  float \_m33 = ( ( ( m31 \* v.m13 ) + ( m32 \* v.m23 ) ) + ( m33 \* v.m33 ) ) + ( m34 \* v.m43 );  float \_m34 = ( ( ( m31 \* v.m14 ) + ( m32 \* v.m24 ) ) + ( m33 \* v.m34 ) ) + ( m34 \* v.m44 );  float \_m41 = ( ( ( m41 \* v.m11 ) + ( m42 \* v.m21 ) ) + ( m43 \* v.m31 ) ) + ( m44 \* v.m41 );  float \_m42 = ( ( ( m41 \* v.m12 ) + ( m42 \* v.m22 ) ) + ( m43 \* v.m32 ) ) + ( m44 \* v.m42 );  float \_m43 = ( ( ( m41 \* v.m13 ) + ( m42 \* v.m23 ) ) + ( m43 \* v.m33 ) ) + ( m44 \* v.m43 );  float \_m44 = ( ( ( m41 \* v.m14 ) + ( m42 \* v.m24 ) ) + ( m43 \* v.m34 ) ) + ( m44 \* v.m44 );  return Matrix4x4 ( \_m11, \_m12, \_m13, \_m14, \_m21, \_m22, \_m23, \_m24, \_m31, \_m32, \_m33, \_m34, \_m41, \_m42, \_m43, \_m44 );  }  Matrix4x4 Matrix4x4::operator\* ( float v ) {  return Matrix4x4 ( m11 \* v, m12 \* v, m13 \* v, m14 \* v, m21 \* v, m22 \* v, m23 \* v, m24 \* v, m31 \* v, m32 \* v, m33 \* v, m34 \* v, m41 \* v, m42 \* v, m43 \* v, m44 \* v );  }  Matrix4x4 Matrix4x4::operator/ ( Matrix4x4 & v ) {  return Matrix4x4 ( m11 / v.m11, m12 / v.m12, m13 / v.m13, m14 / v.m14, m21 / v.m21, m22 / v.m22, m23 / v.m23, m24 / v.m24, m31 / v.m31, m32 / v.m32, m33 / v.m33, m34 / v.m34, m41 / v.m41, m42 / v.m42, m43 / v.m43, m44 / v.m44 );  }  Matrix4x4 Matrix4x4::operator/ ( float v ) {  return Matrix4x4 ( m11 / v, m12 / v, m13 / v, m14 / v, m21 / v, m22 / v, m23 / v, m24 / v, m31 / v, m32 / v, m33 / v, m34 / v, m41 / v, m42 / v, m43 / v, m44 / v );  }  bool Matrix4x4::operator== ( Matrix4x4 & v ) {  return m11 == v.m11 && m12 == v.m12 && m13 == v.m13 && m14 == v.m14 && m21 == v.m21 && m22 == v.m22 && m23 == v.m23 && m24 == v.m24 && m31 == v.m31 && m32 == v.m32 && m33 == v.m33 && m34 == v.m34 && m41 == v.m41 && m42 == v.m42 && m43 == v.m43 && m44 == v.m44;  }  Matrix4x4 Matrix4x4::transpose () {  return Matrix4x4 ( m11, m21, m31, m41, m12, m22, m32, m42, m13, m23, m33, m43, m14, m24, m34, m44 );  }  Matrix4x4 Matrix4x4::invert () {  float det1 = m11 \* m22 - m12 \* m21;  float det2 = m11 \* m23 - m13 \* m21;  float det3 = m11 \* m24 - m14 \* m21;  float det4 = m12 \* m23 - m13 \* m22;  float det5 = m12 \* m24 - m14 \* m22;  float det6 = m13 \* m24 - m14 \* m23;  float det7 = m31 \* m42 - m32 \* m41;  float det8 = m31 \* m43 - m33 \* m41;  float det9 = m31 \* m44 - m34 \* m41;  float det10 = m32 \* m43 - m33 \* m42;  float det11 = m32 \* m44 - m34 \* m42;  float det12 = m33 \* m44 - m34 \* m43;  float detMatrix = ( float ) ( det1 \* det12 - det2 \* det11 + det3 \* det10 + det4 \* det9 - det5 \* det8 + det6 \* det7 );  float invDetMatrix = 1.0f / detMatrix;  return Matrix4x4 ( m11, m12, m13, m14, m21, m22, m23, m24, m31, m32, m33, m34, m41, m42, m43, m44 ) \* invDetMatrix;  }  float Matrix4x4::determinant () {  float num22 = m11, num21 = m12, num20 = m13, num19 = m14;  float num12 = m21, num11 = m22, num10 = m23, num9 = m24;  float num8 = m31, num7 = m32, num6 = m33, num5 = m34;  float num4 = m41, num3 = m42, num2 = m43, num1 = m44;  float num18 = ( num6 \* num1 ) - ( num5 \* num2 ), num17 = ( num7 \* num1 ) - ( num5 \* num3 );  float num16 = ( num7 \* num2 ) - ( num6 \* num3 ), num15 = ( num8 \* num1 ) - ( num5 \* num4 );  float num14 = ( num8 \* num2 ) - ( num6 \* num4 ), num13 = ( num8 \* num3 ) - ( num7 \* num4 );  return ( ( ( ( num22 \* ( ( ( num11 \* num18 ) - ( num10 \* num17 ) ) + ( num9 \* num16 ) ) ) -  ( num21 \* ( ( ( num12 \* num18 ) - ( num10 \* num15 ) ) + ( num9 \* num14 ) ) ) ) +  ( num20 \* ( ( ( num12 \* num17 ) - ( num11 \* num15 ) ) + ( num9 \* num13 ) ) ) ) -  ( num19 \* ( ( ( num12 \* num16 ) - ( num11 \* num14 ) ) + ( num10 \* num13 ) ) ) );  }  Matrix4x4 Matrix4x4::identity () {  return Matrix4x4 ( 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 );  }  Matrix4x4 Matrix4x4::translate ( Vector3 & translate ) {  return Matrix4x4 ( 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, translate.x, translate.y, translate.z, 1 );  }  Matrix4x4 Matrix4x4::scale ( Vector3 & scale )  {  return Matrix4x4 ( scale.x, 0, 0, 0, 0, scale.y, 0, 0, 0, 0, scale.z, 0, 0, 0, 0, 1 );  }  Matrix4x4 Matrix4x4::rotateX ( float r ) {  return Matrix4x4 ( 1, 0, 0, 0, 0, cosf ( r ), sinf ( r ), 0, 0, -sinf ( r ), cosf ( r ), 0, 0, 0, 0, 1 );  }  Matrix4x4 Matrix4x4::rotateY ( float r ) {  return Matrix4x4 ( cosf ( r ), 0, -sinf ( r ), 0, 0, 1, 0, 0, sinf ( r ), 0, cosf ( r ), 0, 0, 0, 0, 1 );  }  Matrix4x4 Matrix4x4::rotateZ ( float r ) {  return Matrix4x4 ( cosf ( r ), sinf ( r ), 0, 0, -sinf ( r ), cosf ( r ), 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 );  }  Matrix4x4 Matrix4x4::orthographicOffCenterLH ( float l, float r, float b, float t, float zn, float zf ) {  return Matrix4x4 ( 2 / ( r - l ), 0, 0, 0, 0, 2 / ( t - b ), 0, 0, 0, 0, 1 / ( zf - zn ), 0, ( l + r ) / ( l - r ), ( t + b ) / ( b - t ), -zn / ( zf - zn ), 1 );  }  Matrix4x4 Matrix4x4::orthographicOffCenterRH ( float l, float r, float b, float t, float zn, float zf ) {  return Matrix4x4 ( 2 / ( r - l ), 0, 0, 0, 0, 2 / ( t - b ), 0, 0, 0, 0, 1 / ( zn - zf ), 0, ( l + r ) / ( l - r ), ( t + b ) / ( b - t ), zn / ( zf - zn ), 1 );  }  Matrix4x4 Matrix4x4::perspectiveFieldOfViewLH ( float fov, float aspect, float zn, float zf ) {  float yScale = ( float ) ( cosf ( fov / 2 ) / sinf ( fov / 2 ) ), xScale = yScale / aspect;  return Matrix4x4 ( xScale, 0, 0, 0, 0, yScale, 0, 0, 0, 0, zf / ( zf - zn ), -1, 0, 0, -zn \* zf / ( zf - zn ), 0 );  }  Matrix4x4 Matrix4x4::perspectiveFieldOfViewRH ( float fov, float aspect, float zn, float zf ) {  float yScale = ( float ) ( cosf ( fov / 2 ) / sinf ( fov / 2 ) ), xScale = yScale / aspect;  return Matrix4x4 ( xScale, 0, 0, 0, 0, yScale, 0, 0, 0, 0, zf / ( zn - zf ), -1, 0, 0, zn \* zf / ( zn - zf ), 0 );  }  Matrix4x4 Matrix4x4::lookAtLH ( Vector3 & eye, Vector3 & at, Vector3 & up ) {  Vector3 zaxis = ( at - eye ).normalize ();  Vector3 xaxis = up.cross ( zaxis ).normalize ();  Vector3 yaxis = zaxis.cross ( xaxis );  return Matrix4x4 ( xaxis.x, yaxis.x, zaxis.x, 0, xaxis.y, yaxis.y, zaxis.y, 0, xaxis.z, yaxis.z, zaxis.z, 0, -xaxis.dot ( eye ), -yaxis.dot ( eye ), -zaxis.dot ( eye ), 1 );  }  Matrix4x4 Matrix4x4::lookAtRH ( Vector3 & eye, Vector3 & at, Vector3 & up ) {  Vector3 zaxis = ( eye - at ).normalize ();  Vector3 xaxis = up.cross ( zaxis ).normalize ();  Vector3 yaxis = zaxis.cross ( xaxis );  return Matrix4x4 ( xaxis.x, yaxis.x, zaxis.x, 0, xaxis.y, yaxis.y, zaxis.y, 0, xaxis.z, yaxis.z, zaxis.z, 0, -xaxis.dot ( eye ), -yaxis.dot ( eye ), -zaxis.dot ( eye ), 1 );  }  struct StackNode {  daram::Matrix4x4 matrix;  StackNode \* next;  };  StackNode \* modelViewStack;  StackNode \* projectionStack;  Vector4 runnedVertex;  daram::MatrixMode currentMatrixMode;  void daram::mySetMatrixMode(MatrixMode matrixMode) {  currentMatrixMode = matrixMode;  switch (matrixMode) {  case MatrixMode\_ModelView:  if (modelViewStack == nullptr) {  modelViewStack = new StackNode;  modelViewStack->next = nullptr;  }  break;  case MatrixMode\_Projection:  if (projectionStack == nullptr) {  projectionStack = new StackNode;  projectionStack->next = nullptr;  }  break;  }  }  void daram::myLoadIdentity() {  Matrix4x4 identity = Matrix4x4::identity ();  switch (currentMatrixMode) {  case MatrixMode\_ModelView:  modelViewStack->matrix = identity;  break;  case MatrixMode\_Projection:  projectionStack->matrix = identity;  break;  }  }  void daram::myPushMatrix() {  StackNode \* node = new StackNode;  switch (currentMatrixMode) {  case MatrixMode\_ModelView:  node->matrix = modelViewStack->matrix;  node->next = modelViewStack;  modelViewStack = node;  break;  case MatrixMode\_Projection:  node->matrix = projectionStack->matrix;  node->next = projectionStack;  projectionStack = node;  break;  }  }  void daram::myPopMatrix() {  StackNode \* topNode;  switch (currentMatrixMode) {  case MatrixMode\_ModelView:  topNode = modelViewStack;  break;  case MatrixMode\_Projection:  topNode = projectionStack;  break;  default: return;  }  if (topNode->next == nullptr) return;  StackNode \* node = topNode->next;  delete topNode;  switch (currentMatrixMode) {  case MatrixMode\_ModelView: modelViewStack = node; break;  case MatrixMode\_Projection: projectionStack = node; break;  default: return;  }  }  void daram::myMatrixMul(Matrix4x4 & matrix) {  StackNode \* topNode;  switch (currentMatrixMode) {  case MatrixMode\_ModelView: topNode = modelViewStack; break;  case MatrixMode\_Projection: topNode = projectionStack; break;  default: return;  }  Matrix4x4 top = topNode->matrix;  topNode->matrix = top \* matrix;  }  void daram::myRunMatrix(Vector3 & vertex) {  Vector4 v(vertex, 1);  if ( modelViewStack ) v = v \* modelViewStack->matrix;  if ( projectionStack ) v = v \* projectionStack->matrix;  runnedVertex = v;  }  void daram::myTranslateX(float tx) {  Vector3 t(tx, 0, 0);  Matrix4x4 translate = Matrix4x4::translate(t);  myMatrixMul(translate);  }  void daram::myTranslateY(float ty) {  Vector3 t(0, ty, 0);  Matrix4x4 translate = Matrix4x4::translate(t);  myMatrixMul(translate);  }  void daram::myTranslateZ(float tz) {  Vector3 t(0, 0, tz);  Matrix4x4 translate = Matrix4x4::translate(t);  myMatrixMul(translate);  }  #define ANGLE2RAD(x) x / 180.0f \* 3.141592f  void daram::myRotateX(float rx) {  Matrix4x4 rotate = Matrix4x4::rotateX ( ANGLE2RAD ( rx ) );  myMatrixMul(rotate);  }  void daram::myRotateY(float ry) {  Matrix4x4 rotate = Matrix4x4::rotateY ( ANGLE2RAD ( ry) );  myMatrixMul(rotate);  }  void daram::myRotateZ(float rz) {  Matrix4x4 rotate = Matrix4x4::rotateZ ( ANGLE2RAD ( rz) );  myMatrixMul(rotate);  }  void daram::myScaleX(float sx) {  Vector3 s(sx, 0, 0);  Matrix4x4 scale = Matrix4x4::scale(s);  myMatrixMul(scale);  }  void daram::myScaleY(float sy) {  Vector3 s(0, sy, 0);  Matrix4x4 scale = Matrix4x4::scale(s);  myMatrixMul(scale);  }  void daram::myScaleZ(float sz) {  Vector3 s(0, 0, sz);  Matrix4x4 scale = Matrix4x4::scale(s);  myMatrixMul(scale);  }  void daram::myLookAt(Vector3 & eye, Vector3 & at, Vector3 & up) {  Matrix4x4 matrix = Matrix4x4::lookAtRH(eye, at, up);  myMatrixMul(matrix);  }  Matrix4x4 daram::getTopMatrix() {  StackNode \* topNode;  switch (currentMatrixMode) {  case MatrixMode\_ModelView: topNode = modelViewStack; break;  case MatrixMode\_Projection: topNode = projectionStack; break;  default: return Matrix4x4 ();  }  return topNode->matrix;  }  Vector4 daram::getRunnedVertex() { return runnedVertex; }  #include <iostream>  void daram::showMatrix4x4(Matrix4x4 & matrix) {  float \* arr = (float\*)&(matrix = matrix.transpose());  for(int y = 0; y < 4; ++y) {  for(int x = 0; x < 4; ++x) std::cout << arr[y \* 4 + x] << " ";  std::cout << std::endl;  }  }  void daram::showVector4(Vector4 & vector) {  float \* arr = (float\*)&vector;  for(int x = 0; x < 4; ++x) std::cout << arr[x] << " ";  std::cout << std::endl;  } |
| ***main.cpp (코드 테스트용 파일)*** |
| #include <iostream>  #include "MyMatrix.h"  int main(int argc, const char \* argv[])  {  daram::mySetMatrixMode ( daram::MatrixMode::MatrixMode\_ModelView );  daram::myLoadIdentity ();  std::cout << "-- MatrixMode - ModelView, Loaded Identity --" << std::endl;  daram::showMatrix4x4 ( daram::getTopMatrix () );  daram::myLookAt ( daram::Vector3 ( 20, 20, 20 ), daram::Vector3 ( 0, 0, 0 ), daram::Vector3 ( 0, 1, 0 ) );  std::cout << "-- MatrixMode - ModelView, LookAt --" << std::endl;  daram::showMatrix4x4 ( daram::getTopMatrix () );  daram::myTranslateX ( 10 );  daram::myTranslateY ( 20 );  std::cout << "-- Translate Tx(10), Ty(20) --" << std::endl;  daram::showMatrix4x4 ( daram::getTopMatrix () );  daram::myPushMatrix ();  std::cout << "-- PushMatrix --" << std::endl;  daram::showMatrix4x4 ( daram::getTopMatrix () );  daram::myRotateX ( 128 );  daram::myRotateZ ( 65 );  std::cout << "-- Rotate Rx(128), Rz(65) --" << std::endl;  daram::showMatrix4x4 ( daram::getTopMatrix () );  daram::myRunMatrix ( daram::Vector3 ( 20, 20, 20 ) );  std::cout << "-- Run Vertex(20, 20, 20) --" << std::endl;  daram::showVector4 ( daram::getRunnedVertex () );  daram::myPopMatrix ();  std::cout << "-- PopMatrix --" << std::endl;  daram::showMatrix4x4 ( daram::getTopMatrix () );  daram::myRunMatrix ( daram::Vector3 ( 20, 20, 20 ) );  std::cout << "-- Run Vertex(20, 20, 20) --" << std::endl;  daram::showVector4 ( daram::getRunnedVertex () );    return 0;  } |

Vector2/3/4와 Matrix4x4는 기존에 게임 제작을 위해 직접 만들었던 클래스를 재활용했으며, 행렬 스택은 링크드 스택 방식으로 구현했습니다.

행렬 구조 등은 MSDN의 DirectX9을 참고했으며, 따라서 컴퓨터 그래픽스 시간에 표기하는 방식의 전치 형태를 가지기 때문에 행렬 출력 시에는 전치한 후 출력하였습니다.

모든 코드는 C++로 작성했으며, 이번 과제에 있어서 가장 중요한 코드는 클래스에 속해있지 않은 함수들입니다.

# 결과



# 논의

## 이 Lab의 키는 무엇인가?

이번 Lab은 기존의 OpenGL 행렬 스택을 사용하지 않고 직접 행렬 스택을 구현해보는 것이 키였습니다.

## 무슨 실수를 하고, 무엇을 배웠는가?

따로 실수한 것은 없었습니다.

## 프로그램을 어떻게 향상시킬 수 있겠는가?

C 형식의 행렬 스택 방식이 아닌 C++에서의 객체 방식 행렬 스택을 구현하면 더 사용하기 쉽고 편할 것이라고 생각됩니다.