Lab #6 – myLighTing

Computer Graphics

컴퓨터 그래픽스

남병규 교수님

201002506 진재연

2014. 11. 26

# Lab의 목적

이번 과제의 목적은 OpenGL의 Lighting의 API를 본떠 라이팅 시스템을 직접 구현해보는 것입니다.

# 소스 코드

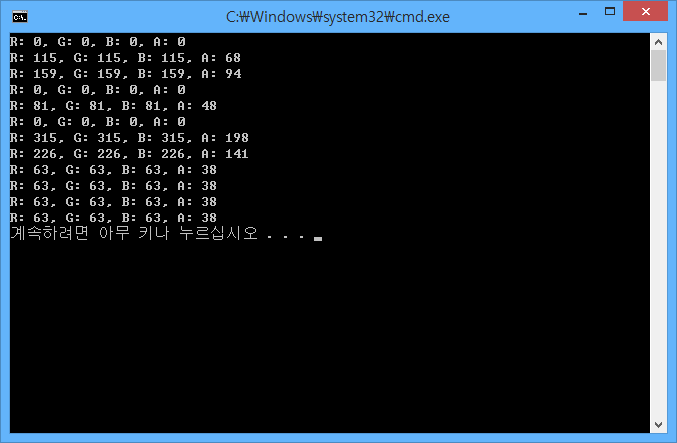
|  |
| --- |
| ***MyMatrix.h*** |
| //  // MyMatrix.h  // MatrixLib  //  // Created by 진재연 on 2014. 10. 15..  // Copyright (c) 2014년 DARAM WORLD. All rights reserved.  //  #ifndef \_\_MatrixLib\_\_MyMatrix\_\_  #define \_\_MatrixLib\_\_MyMatrix\_\_  namespace daram {  struct Vector2;  struct Vector3;  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- ( float v );  Vector3 operator\* ( Vector3 v );  Vector3 operator\* ( float v );  Vector3 operator/ ( Vector3 v );  Vector3 operator/ ( float 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- ( float v );  Vector4 operator- ( );  Vector4 operator\* ( Vector4 v );  Vector4 operator\* ( float v );  Vector4 operator\* ( Matrix4x4 matrix );  Vector4 operator/ ( Vector4 v );  Vector4 operator/ ( float 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 );    static Matrix4x4 rotatePitchYawRoll ( float yaw, float pitch, float roll );    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);    void myFrustum ( int l, int r, int b, int t, float n, float f );  void myOrtho ( int l, int r, int b, int t, float n, float f );    Matrix4x4 getTopMatrix();  Vector4 getRunnedVertex();    void showMatrix4x4(Matrix4x4 matrix);  void showVector4(Vector4 vector);  }  #endif |
| ***MyMatrix.cpp*** |
| //  // MyMatrix.cpp  // MatrixLib  //  // Created by 진재연 on 2014. 10. 15..  // Copyright (c) 2014년 DARAM WORLD. All rights reserved.  //  #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- ( float v ) { return Vector3 ( x - v, y - v, z - v ); }  Vector3 Vector3::operator\* ( Vector3 v ) { return Vector3 ( x \* v.x, y \* v.y, z \* v.z ); }  Vector3 Vector3::operator\* ( float v ) { return Vector3 ( x \* v, y \* v, z \* v ); }  Vector3 Vector3::operator/ ( Vector3 v ) { return Vector3 ( x / v.x, y / v.y, z / v.z ); }  Vector3 Vector3::operator/ ( float v ) { return Vector3 ( x / v, y / v, z / v ); }  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::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- ( float v ) { return Vector4 ( x - v, y - v, z - v, w - v ); }  Vector4 Vector4::operator\* ( float v ) { return Vector4 ( x \* v, y \* v, z \* v, w \* v ); }  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 ); }  Vector4 Vector4::operator/ ( float v ) { return Vector4 ( x / v, y / v, z / v, w / v); }  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 (); if ( len != 0 ) return Vector4 ( x / len, y / len, z / len, w / len ); return Vector4 (); }  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);  }  void daram::myFrustum ( int l, int r, int b, int t, float n, float f )  {  Matrix4x4 matrix (  (2 \* n) / (r - l), 0, (r + l) / (r - l), 0,  0, (2 \* n) / (t - b), (t + b) / (t - b), 0,  0, 0, - ((f + n) / (f - n)), -((2 \* f \* n) / (f - n)),  0, 0, -1, 0  );  matrix = matrix.transpose();  myMatrixMul(matrix);  }  void daram::myOrtho ( int l, int r, int b, int t, float n, float f )  {  Matrix4x4 matrix = Matrix4x4::orthographicOffCenterRH(l, r, b, t, n, f);  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;  } |
| ***MyLighting.h (라이팅 구현 파일)*** |
| //  // MyLigting.h  // MyGL  //  // Created by jinjae-yeon on 2014. 12. 2..  // Copyright (c) 2014년 DARAM WORLD. All rights reserved.  //  #ifndef \_\_MyGL\_\_MyLigting\_\_  #define \_\_MyGL\_\_MyLigting\_\_  #include "MyMatrix.h"  namespace daram {  enum LightProperty  {  LightProperty\_Ambient,  LightProperty\_Diffuse,  LightProperty\_Specular,  LightProperty\_Position,  };    struct Vertex  {  Vector3 position;  Vector3 normal;  };    void mySetStreamSource ( Vertex \* vertices, int length );    void myLightEnable ( int id, bool enable );  void mySetLight ( int id, LightProperty prop, float \* coef );    Vector4 \* myLigting ( int id );  }  #endif /\* defined(\_\_MyGL\_\_MyLigting\_\_) \*/ |
| ***MyLighting.cpp (라이팅 구현 파일)*** |
| //  // MyLigting.cpp  // MyGL  //  // Created by jinjae-yeon on 2014. 12. 2..  // Copyright (c) 2014년 DARAM WORLD. All rights reserved.  //  #include "MyLigting.h"  #include <cstring>  #include <cmath>  daram::Vector4 arrToVec4 ( float \* arr ) { return daram::Vector4 ( arr [ 0 ], arr [ 1 ], arr [ 2 ], arr [ 3 ] ); }  daram::Vector3 arrToVec3 ( float \* arr ) { return daram::Vector3 ( arr [ 0 ], arr [ 1 ], arr [ 2 ] ); }  float saturate ( float x ) { return ( x > 0 ) ? ( ( x < 1 ) ? x : 1 ) : 0; }  daram::Vertex vertices [ 16 ];  void daram::mySetStreamSource ( daram::Vertex \* \_vertices, int length )  {  memcpy(vertices, \_vertices, sizeof(daram::Vertex) \* length);  }  struct \_Light  {  bool isEnabled;  daram::Vector4 ambient;  daram::Vector4 diffuse;  daram::Vector4 specular;  daram::Vector3 position;  } lights [ 8 ];  daram::Vector4 resultColor [ 16 ];  void daram::myLightEnable ( int id, bool enable )  {  lights [ id ].isEnabled = enable;  }  void daram::mySetLight ( int id, daram::LightProperty prop, float \* coef )  {  \_Light \* light = &lights [ id ];  switch (prop) {  case LightProperty\_Ambient: light->ambient = arrToVec4(coef); break;  case LightProperty\_Diffuse: light->diffuse = arrToVec4 ( coef ); break;  case LightProperty\_Specular: light->specular = arrToVec4 ( coef ); break;  case LightProperty\_Position: light->position = arrToVec3(coef); break;    default: break;  }  }  daram::Vector4 \* daram::myLigting ( int id )  {  if ( !lights [ id ].isEnabled )  return nullptr;    for ( int i = 0; i < 16; ++i )  {  daram::Vector3 viewDir = daram::Vector3( 5, 5, 0 ) - vertices [ i ].position;  daram::Vector3 lightDir = ( lights [ id ].position - vertices [ i ].position ) / 15;  float ldDot = lightDir.dot ( lightDir );  float attenuetion = saturate ( 1.0f - ldDot );    daram::Vector3 n = vertices [ i ].normal.normalize ();  daram::Vector3 l = lightDir.normalize ();  daram::Vector3 v = viewDir.normalize ();  daram::Vector3 h = ( l + v ).normalize ();    float nDotL = saturate ( n.dot ( l ) );  float nDotH = saturate ( n.dot ( h ) );  float power = ( nDotL == 0.0f ) ? 0.0f : pow ( nDotH, 1 );    resultColor [ i ] = ( lights [ id ].ambient \* attenuetion +  ( lights [ id ].diffuse \* nDotL \* attenuetion ) +  ( lights [ id ].specular \* power \* attenuetion ) );  }    return resultColor;  } |
| ***main.cpp (코드 테스트용 파일)*** |
| //  // main.cpp  // MatrixLib  //  // Created by 진재연 on 2014. 10. 15..  // Copyright (c) 2014년 DARAM WORLD. All rights reserved.  //  #include <iostream>  #include "MyMatrix.h"  #include "MyLigting.h"  int main(int argc, const char \* argv[])  {  daram::Vertex vertices [ 12 ] = {  { daram::Vector3 ( -2.0f, 0.0f, -2.0f ), daram::Vector3 ( 0.0f, 0.707f, -0.707f ) },  { daram::Vector3 ( 0.0f, 1.0f, 0.0f ), daram::Vector3 ( 0.0f, 0.707f, -0.707f ) },  { daram::Vector3 ( 2.0f, 0.0f, -2.0f ), daram::Vector3 ( 0.0f, 0.707f, -0.707f ) },    { daram::Vector3 ( -2.0f, 0.0f, 2.0f ), daram::Vector3 ( -0.707f, 0.707f, 0.0f ) },  { daram::Vector3 ( 0.0f, 2.0f, 0.0f ), daram::Vector3 ( -0.707f, 0.707f, 0.0f ) },  { daram::Vector3 ( -2.0f, 0.0f, -2.0f ), daram::Vector3 ( -0.707f, 0.707f, 0.0f ) },    { daram::Vector3 ( 2.0f, 0.0f, -2.0f ), daram::Vector3 ( 0.707f, 0.707f, 0.0f ) },  { daram::Vector3 ( 0.0f, 2.0f, 0.0f ), daram::Vector3 ( 0.707f, 0.707f, 0.0f ) },  { daram::Vector3 ( 2.0f, 0.0f, 2.0f ), daram::Vector3 ( 0.707f, 0.707f, 0.0f ) },    { daram::Vector3 ( 2.0f, 0.0f, 2.0f ), daram::Vector3 ( 0.0f, 0.707f, 0.707f ) },  { daram::Vector3 ( 0.0f, 2.0f, 0.0f ), daram::Vector3 ( 0.0f, 0.707f, 0.707f ) },  { daram::Vector3 ( -2.0f, 0.0f, 2.0f ), daram::Vector3 ( 0.0f, 0.707f, 0.707f ) },  };  daram::mySetStreamSource(vertices, 8);    daram::myLightEnable(0, true);    float pos [] = { 12, 5, 0 };  daram::mySetLight ( 0, daram::LightProperty\_Position, pos );  float amb [] = { 1, 1, 1, 0.6f };  daram::mySetLight ( 0, daram::LightProperty\_Ambient, amb );  float dif [] = { 1, 1, 1, 1 };  daram::mySetLight ( 0, daram::LightProperty\_Diffuse, dif );  float spe [] = { 1, 1, 1, 0.3f };  daram::mySetLight ( 0, daram::LightProperty\_Specular, spe );    daram::Vector4 \* result = daram::myLigting(0);  for ( int i = 0; i < 12; ++i )  printf( "R: %d, G: %d, B: %d, A: %d\n", (int)(result [ i ].x \* 255), (int)(result [ i ].y \* 255), (int)(result [ i ].z \* 255), (int)(result [ i ].w \* 255) );    return 0;  } |

과제에서는 각 연산(Ambient, Diffuse, Specular, Attenuation)을 따로 계산하도록 함수를 만들고, 이를 myLighting에서 가져다 사용할 것을 요구하는 것 같지만, 꼭 이를 따르지 않아도 될 것 같아서 myLighting에 모두 작성하였습니다. 또한 Emission의 구현 방법은 자료에 나와있지 않아 작성하지 않았습니다.

Vector 부분에서 추가적인 연산 함수가 필요하여 추가 구현하였고(벡터의 사착연산에 스칼라 값 계산하는 함수), 주로 작성된 것은 myLighting.h 및 cpp 파일입니다.

테스트는 피라미드 모양의 정점 배열을 이용하여 수행하였고, 이에 대한 빛 계산을 수행하도록 하였습니다. 255의 범위를 넘는 값은 어떻게 처리해야 할지 모르겠어 일단 두었습니다.

# 결과



# 논의

## 이 Lab의 키는 무엇인가?

이번 Lab은 기존의 OpenGL 라이팅을 사용하지 않고 직접 라이팅을 구현해보는 과제로, 광원에 대한 정점의 라이팅 색상을 구하는 기능을 구현하는 것이 키였습니다.

## 무슨 실수를 하고, 무엇을 배웠는가?

강의 자료가 난해하여 3주간 구현하지 못했고, 결국 라이팅이 구현된 각종 HLSL 및 GLSL 쉐이더 코드를 분석하여 구현하였습니다. 해당 쉐이더들은 모두 정점 쉐이더가 아닌 픽셀 쉐이더에서 구현된 것이었기 때문에 정점에 대해서만 처리하도록 고치는 과정에서 값이 제대로 나오지 않았으며, 현재도 제대로 나오고 있는 것 같지는 않습니다.

## 프로그램을 어떻게 향상시킬 수 있겠는가?

현재는 광원 데이터를 하나만 이용하여 처리하도록 했지만, 추가적으로 하나의 물체에 대해 여러 광원에 대한 처리를 하도록 작성한다면 좀 더 다양한 결과가 나올 수도 있을 것입니다.