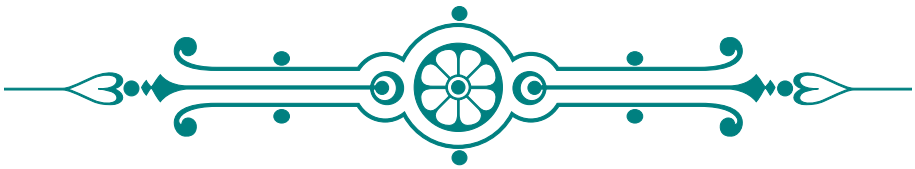

컴퓨터 그래픽스 입문

학번 : 2016110056
학과 : 불교학부
이름 : 박승원
날짜 : 2017년 3월 16일

1. Draw a smiley as pretty as you can. You may google 'smiley face' for more an better references.(5)

Scoring guide

- Draw a color filled face : 1 point
- Draw a smiley mouth : 1 point
- Draw a eyes(left + right) : 2 point
- How much pretty smiley face : 1 point

2. Move the smiley by up/down/left/right key inputs. (5)

Scoring guide

- move left : 1 point
- move right : 1 point
- move up : 1 point
- move down : 1 point
- how well did you write the report : 1 point

원을 그리기 위한 클래스로 원위의 점들을 벡터로 가진다.

```
#include<vector>
#include"matrix.h"

class Circle
{
public:
    Circle(float* ctr, float r, int points_count = 100)
        : center{ctr[0], ctr[1], ctr[2]}, radius{r} {
        Matrix<float> p{r, 0, 0};
        Matrix<float> rz{4, 4}, tr{4, 4};
        rz.glrotateZ(2 * M_PI / points_count);
        tr.gltranslate(ctr[0], ctr[1], ctr[2]);
        for(int i=0; i<points_count; i++) {
            pts.push_back(p);
            p = rz * p;
        }
        for(auto& a : pts) a = tr * a;
    }

    auto begin() {
        return pts.begin();
    }
    auto end() {
        return pts.end();
    }
```

```
    float* ctr() {return center.data();}

protected:
    Matrix<float> center;
    std::vector<Matrix<float>> pts;
    float radius;
};
```

기본 매트릭스, 위치 이동과 기본 자료형으로 쓰임.

```
#pragma once
#include<cstring>
#include<sstream>
#include<cmath>
#include<cassert>
#include<iostream>
#include<iomanip>
#include"combi.h"

template <typename T> class Matrix
{
public:
    Matrix(unsigned short w, unsigned short h) {
        width = w; height = h;
        arr = new T[h * w];
        for(int i=0; i<w * h; i++) arr[i] = 0;
    }

    Matrix(T x, T y, T z, T w = 1) : Matrix{1,4} {
        arr[0] = x; arr[1] = y; arr[2] = z; arr[3] = w;
    }

    Matrix(std::initializer_list<std::initializer_list<T>> li)
        : Matrix<T>{static_cast<unsigned short>(li.begin()->size()),
                    static_cast<unsigned short>(li.size())} {
        int x = 1, y = 1;
        for(auto& a : li) {
            for(auto& b : a) (*this)[x++][y] = b;
            y++; x = 1;
        }
    }

    Matrix(const Matrix<T>& r) : Matrix(r.width, r.height) {
```

```
    for(int i=0; i<width * height; i++) arr[i] = r.arr[i];
}

Matrix(Matrix<T>&& r) {
    arr = r.arr; r.arr = nullptr;
    width = r.width; height = r.height;
}

virtual ~Matrix() {if(arr) delete [] arr;}

/// getters
T* data() const {return arr;}
unsigned short get_width() const{return width;}
unsigned short get_height() const{return height;}

/// operator overloading
T* operator[](int x) {// start from 11 21 31
    assert(x > 0);
    return arr + (x -1) * height - 1;
}
T* operator[](int x)const {// start from 11 21 31
    assert(x > 0);
    return arr + (x -1) * height - 1;
}

Matrix<T> operator+(const Matrix<T>& r) const {
    if(width != r.width || height != r.height) throw "Matrix size not match";
    Matrix<T> m(width, height);
    for(int i=0; i<width*height; i++) m.arr[i] = arr[i] + r.arr[i];
    return m;
}

Matrix<T> operator-(const Matrix<T>& r) const {
    if(width != r.width || height != r.height) throw "Matrix size not match";
    Matrix<T> m(width, height);
    for(int i=0; i<width*height; i++) m.arr[i] = arr[i] - r.arr[i];
    return m;
}

Matrix<T> operator*(const Matrix<T>& r) const {
    if(width != r.height) throw "Matrix size not match";
    Matrix<T> m(r.width, height);
```

```
    for(int x = 1; x <= r.width; x++) for(int y = 1; y <= height; y++)
        m[x][y] = inner_product(y, r.column(x));
    return m;
}

Matrix<T>& operator=(const Matrix<T>& r) {
    if(width != r.width || height != r.height) throw "Matrix size not match";
    for(int i=0; i<width*height; i++) arr[i] = r.arr[i];
    return *this;
}

Matrix<T>& operator*=(const Matrix<T>& r) {
    *this = *this * r;
    return *this;
}

Matrix<T> operator*(const T& r) const {return r * *this;}

bool operator==(const Matrix<T>& r) const {
    if(width != r.width || height != r.height) return false;
    for(int i=0; i<width*height; i++) if(arr[i] != r.arr[i]) return false;
    return true;
}

friend Matrix<T> operator*(const T l, const Matrix<T>& r) {
    Matrix<T> m(r.width, r.height);
    for(int y=0; y<r.height; y++) for(int x=0; x<r.width; x++)
        m.arr[y*r.width+x] = l * r.arr[y*r.width+x];
    return m;
}

Matrix<T> inverse() const{
    auto a = LU_decompose();
    auto P = a[0], L = a[1], U = a[2];
    Matrix<T> I{width, height};
    for(int i=1; i<=height; i++) {
        Matrix B{1, height}; // divide E into column pieces
        B[1][i] = 1;
        auto Ux = LxB(L, P * B); // Ax = B <==> PAx = PB <==> LUx = PB
        auto x = UxB(U, Ux);
        for(int j=1; j<=height; j++) I[i][j] = x[1][j];
    }
```

```
    return I;
}

Matrix<T> transpose() const{
    Matrix<T> m{height, width};
    for(int x=1; x<=width; x++) for(int y=1; y<=height; y++)
        m[y][x] = (*this)[x][y];
    return m;
}

Matrix<T> E() {
    if(width != height) throw "must be square matrix!";
    for(int x = 1; x <= width; x++) for(int y = 1; y <= height; y++) {
        if(x == y) (*this)[x][y] = 1;
        else (*this)[x][y] = 0;
    }
    return *this;
}
Matrix<T> gltranslate(T x, T y, T z) {
    if(width != 4 || height != 4) throw "should be 4x4";
    E();
    (*this)[4][1] = x;
    (*this)[4][2] = y;
    (*this)[4][3] = z;
    return *this;
}
Matrix<T> glrotateZ(T th) {
    if(width != 4 || height != 4) throw "should be 4x4";
    E();
    (*this)[1][1] = cos(th);
    (*this)[2][1] = -sin(th);
    (*this)[1][2] = sin(th);
    (*this)[2][2] = cos(th);
    return *this;
}
Matrix<T> glrotateX(T th) {
    if(width != 4 || height != 4) throw "should be 4x4";
    E();
    (*this)[2][2] = cos(th);
    (*this)[3][2] = -sin(th);
    (*this)[2][3] = sin(th);
    (*this)[3][3] = cos(th);
```

```
        return *this;
    }
    Matrix<T> glrotateY(T th) {
        if(width != 4 || height != 4) throw "should be 4x4";
        E();
        (*this)[1][1] = cos(th);
        (*this)[3][1] = -sin(th);
        (*this)[1][3] = sin(th);
        (*this)[3][3] = cos(th);
        return *this;
    }
    Matrix<T> glscale(T x, T y, T z) {
        if(width != 4 || height != 4) throw "should be 4x4";
        E();
        (*this)[1][1] = x;
        (*this)[2][2] = y;
        (*this)[3][3] = z;
        return *this;
    }

    Matrix<T> One() const {
        for(int i=0; i<width*height; i++) arr[i] = 1;
    }
    Matrix<T> surround(T wall = 0) const {
        Matrix<T> m{width + 1, height +1};
        for(int i=0; i<m.width*m.height; i++) m.arr[i] = wall;
        for(int x=1; x<=width; x++) for(int y=1; y<=height; y++)
            m[x+1][y+1] = (*this)[x][y];
        return m;
    }

protected:
    T* arr;
    unsigned short width, height;

private:
    T* column(int x) const{
        return arr + (x - 1) * height;
    }

    T inner_product(int row, T* col) const{
        T sum = 0;
```

```
    for(int i=0; i<width; i++) sum += (*this)[i+1][row] * *(col+i);
    return sum;
}
static Matrix<T> LU_decompose(Matrix<T> m) {
    int w = m.width;
    int h = m.height;
    if(!m[1][1]) return Matrix<T>{w,h};
    if(m.width == 1) return m;
    for(int y=2; y<=h; y++) m[1][y] /= m[1][1]; // c /= a11
    for(int x=2; x<=w; x++) for(int y=2; y<=h; y++)
        m[x][y] -= m[x][1] * m[1][y]; // A' -= ch
    Matrix<T> mm{w-1, w-1};
    for(int x=1; x<w; x++) for(int y=1; y<w; y++) mm[x][y] = m[x+1][y+1];
    mm = LU_decompose(mm);// A' part recursive
    if(!mm[1][1]) return Matrix<T>{w,h};//if a11 == 0 -> change P, redo
    for(int x=1; x<w; x++) for(int y=1; y<w; y++) m[x+1][y+1] = mm[x][y];
    return m;
}

auto LU_decompose() const{
    if(width != height) throw "should be square";
    nPr npr{width, width};
    while(npr.next()) {
        Matrix<T> P{width, width};// 조합으로 순열 매트릭스 생성 .
        for(int j=1,i=0; j<=height; j++) P[npr[i++]][j] = 1;
        auto m = P * (*this);
        m = LU_decompose(m);
        if(!m[1][1]) continue;
        else {
            Matrix<T> L{width, height};
            Matrix<T> U{width, height};
            for(int x=1; x<=width; x++)
                for(int y=1; y<=width; y++)
                    if(x<y) L[x][y] = m[x][y];
                    else U[x][y] = m[x][y];
            for(int x=1; x<=width; x++) L[x][x] = 1;
            return std :: array<Matrix<T>, 3>{P, L, U};
        }
    }
    throw "no inverse";
}
static Matrix<T> LxB(Matrix<T> L, Matrix<T> B)
```

```
    {///get x from Lx = B
        int h = L.get_height();
        if(L.get_width() != h || B.get_width() != 1 || B.get_height() != h)
            throw "type mismatch";
        Matrix<T> x{1, h};
        for(int i=1; i<=h; i++) {
            T sum = 0;
            for(int j=1; j<i; j++) sum += L[j][i] * x[1][j];
            x[1][i] = B[1][i] - sum;
        }
        return x;
    }

    static Matrix<T> UxB(Matrix<T> U, Matrix<T> B)
    {///get x from Ux = B
        int h = U.get_height();
        if(U.get_width() != h || B.get_width() != 1 || B.get_height() != h)
            throw "type mismatch";
        Matrix<T> x{1, h};
        for(int i=h; i>0; i--) {
            T sum = 0;
            for(int j=h; j>i; j--) sum += x[1][j] * U[j][i];
            x[1][i] = (B[1][i] - sum) / U[i][i];
        }
        return x;
    }
};

template <typename T> std::ostream& operator<<(std::ostream& o, const Matrix<T>& r){
    int w = r.get_width(), h = r.get_height();
    int gap[w+1] {0,};
    for(int y=1; y<=h; y++) for(int x=1; x<=w; x++) {
        std::stringstream ss;
        ss << r[x][y];
        int sz = ss.str().length();
        if(gap[x] < sz) gap[x] = sz;
    }

    o << "\u23a1" << ' ';
    for(int x=1; x<=w; x++) o << std::setw(gap[x]) << r[x][1] << ' ';
    o << "\u23a4" << std::endl;
    for(int y=2; y<h; y++) {
```

```
        o << "\u23a2" << ' ';
        for(int x=1; x<=w; x++) o << std::setw(gap[x]) << r[x][y] << ' ';
        o << "\u23a5" << std::endl;
    }
    o << "\u23a3" << ' ';
    for(int x=1; x<=w; x++) o << std::setw(gap[x]) << r[x][h] << ' ';
    o << "\u23a6" << std::endl;

    return o;
}

template<typename T> class MatrixStream : public Matrix<T>
{
public:
    MatrixStream(const Matrix<T>& m) : Matrix<T>{m} {
        int w = this->width, h = this->height;
        gap = new int[w];
        memset((void*)gap, 0, sizeof(int) * w);
        linebyline = new std::string[h];
        for(int y=1; y<=h; y++) for(int x=1; x<=w; x++) {
            std::stringstream ss;
            ss << (*this)[x][y];
            int sz = ss.str().length();
            if(gap[x-1] < sz) gap[x-1] = sz; // get maximum length
        } // print with setw

        std::stringstream ss;
        ss << (h-1 ? "\u23a1" : "[ "); // matrix bracket first line
        for(int x=1; x<=w; x++) ss << std::setw(gap[x-1]) << (*this)[x][1] << ' ';
        ss << (h-1 ? "\u23a4" : "]");
        linebyline[0] = ss.str();
        ss.str("");
        ss.clear();

        for(int y=2; y<h; y++) { // middle lines
            ss << "\u23a2" << ' ';
            for(int x=1; x<=w; x++) ss << std::setw(gap[x-1]) << (*this)[x][y] << ' ';
            ss << "\u23a5";
            linebyline[y-1] = ss.str();
            ss.str("");
            ss.clear();
        }
```

```
        if(h > 1) { // last line
            ss << "\u23a3" << ' ';
            for(int x=1; x<=w; x++) ss << std::setw(gap[x-1]) << (*this)[x][h] << ' ';
            ss << "\u23a6";
            linebyline [h-1] = ss.str() ;
        }
    }

    ~MatrixStream() {
        if(gap) delete[] gap;
        if( linebyline ) delete[] linebyline ;
    }

    std :: string space() { // make space string wide exactly the same width of matrix
        std :: string s; // to occupy the blank space
        int sum = 0;
        for(int i=0; i<this->width; i++) sum += gap[i];
        for(int i=0; i < sum + this->width + 3; i++) s += ' ';
        return s;
    }

    template<typename T2>
    friend std :: ostream& operator<<(std::ostream& o, MatrixStream<T2>& r);

protected:
    int* gap = nullptr ; ///< contain biggest output width per every x -> setw()
    std :: string* linebyline = nullptr ; ///< output line by line
    int pos = 0; ///< indicate what line to print
};

template<typename T>
std :: ostream& operator<<(std::ostream& o, MatrixStream<T>& r) {
    if(r.pos == r.height) r.pos = 0;
    o << r.linebyline [r.pos++];
    return o;
}
```

메인 파일

```
//#include <glew.h>
#include"matrix.h"
#include <GLFW/glfw3.h>
```

```
#include <cstring>
#include <stdlib.h>          // srand, rand
#include <thread>            // std :: this_thread :: sleep_for
#include <chrono>            // std :: chrono :: seconds
#include <iostream>
#include<cmath>
#include" glutil .h"
#include" circle .h"
using namespace std;

const int wt = 640;
const int ht = 480;

extern Matrix<float> translate ;
int main(void)
{
    translate . gltranslate (0,0,0) ;
    if (! glfwInit () ) return -1;
    GLFWwindow* window = glfwCreateWindow(wt, ht, "Smiley Face", NULL, NULL);
    if (! glinit (window)) return -1;

    const float theta = 2.0 * M_PI / 20;
    Matrix<float> m{4,4};
    m.glrotateZ ( theta ) ;

    int width, height ;
    glfwGetFramebufferSize(window, &width, &height);
    glViewport(0, 0, width, height ) ;
    glortho (2) ;
    glColor3f (1,0,0) ;

    Matrix<float> center {0,0,0};
    Circle bcircle {center . data () , 1};
    Circle ycircle {center . data () , 0.95};
    float leye [3], reye [3];
    memcpy(leye, center . data () , sizeof( float ) * 3) ;
    memcpy(reye, center . data () , sizeof( float ) * 3) ;
    leye [0] -= 0.3; reye [0] += 0.3;
    leye [1] += 0.5; reye [1] += 0.5;
    Circle leyec{leye , 0.2};
    Circle reyec{reye , 0.2};
    Circle mouth{center.data () , 0.7};
```

```
/* Loop until the user closes the window */
while (!glfwWindowShouldClose(window)) {
    /* Render here */
    glClear(GL_COLOR_BUFFER_BIT);

    //TODO: draw here
    glBegin(GL_TRIANGLE_FAN);
    glVertex3fv( bcircle.ctr() );
    glColor3f (0,0,0) ;
    for(auto& a : bcircle ) {
        a = translate * a;
        glVertex3fv(a.data() );
    }
    glVertex3fv( bcircle.begin()->data());
    glEnd();

    glBegin(GL_TRIANGLE_FAN);
    glColor3f (1,1,0) ;
    for(auto& a : ycircle ) {
        a = translate * a;
        glVertex3fv(a.data() );
    }
    glVertex3fv( ycircle.begin()->data());
    glEnd();

    glBegin(GL_TRIANGLE_FAN);
    glColor3f (0,0,0) ;
    for(auto& a : leyec ) {
        a = translate * a;
        glVertex3fv(a.data() );
    }
    glEnd();

    glBegin(GL_TRIANGLE_FAN);
    for(auto& a : reyec) {
        a = translate * a;
        glVertex3fv(a.data() );
    }
    glEnd();

    glLineWidth(20);
```

```
        glBegin(GL_LINE_STRIP);
        auto it = mouth.begin();
        it += 50;
        for(int i=50; i<99; i++) {
            *it = translate **it;
            glVertex3fv(it++->data());
        }
        glEnd();

        glfwSwapBuffers(window);
        glfwPollEvents(); // glfwWaitEvents();
        std::this_thread::sleep_for(std::chrono::milliseconds(100));
    }

    glfwTerminate();

    return 0;
}
```