제출일 : 2022. 06. 20.

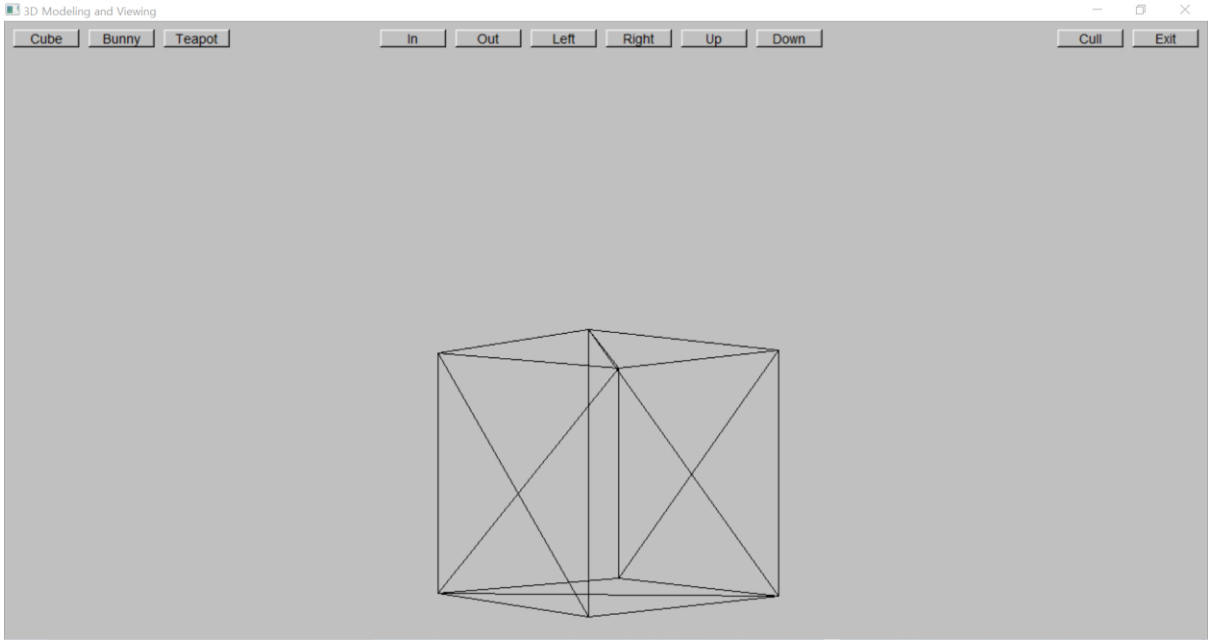
KWANGWOON UNIVERSITY

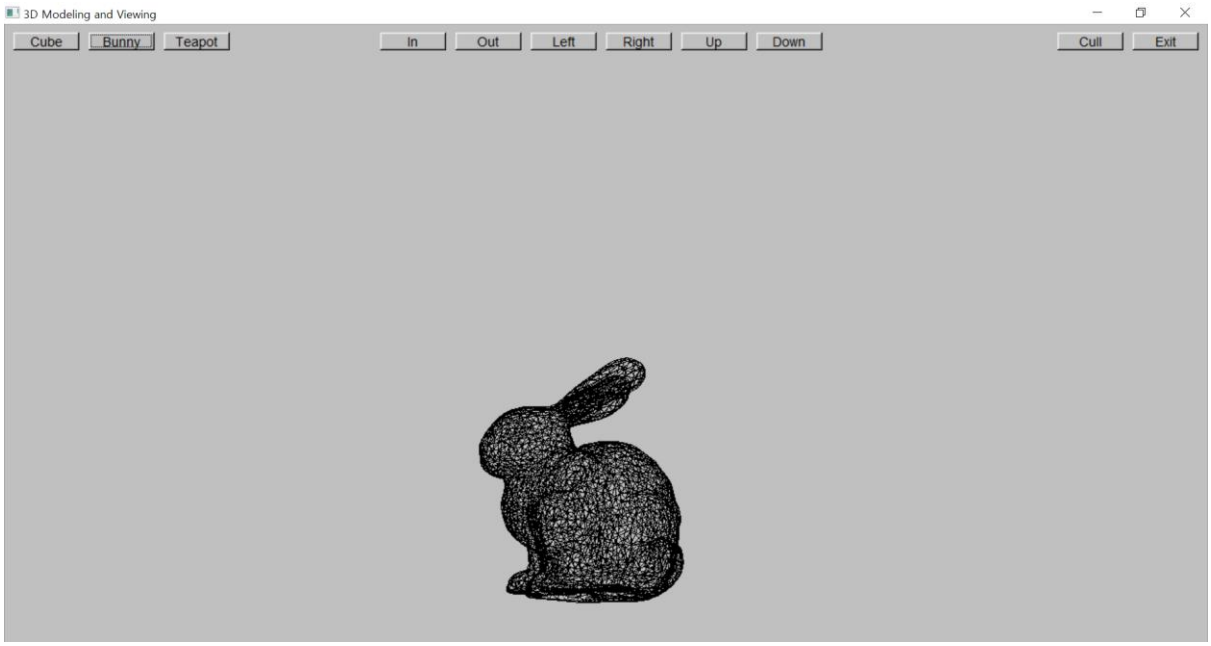
담당교수	이강훈 교수님	학 번	2021203034
학 과	소프트웨어학부	이 름	허찬영

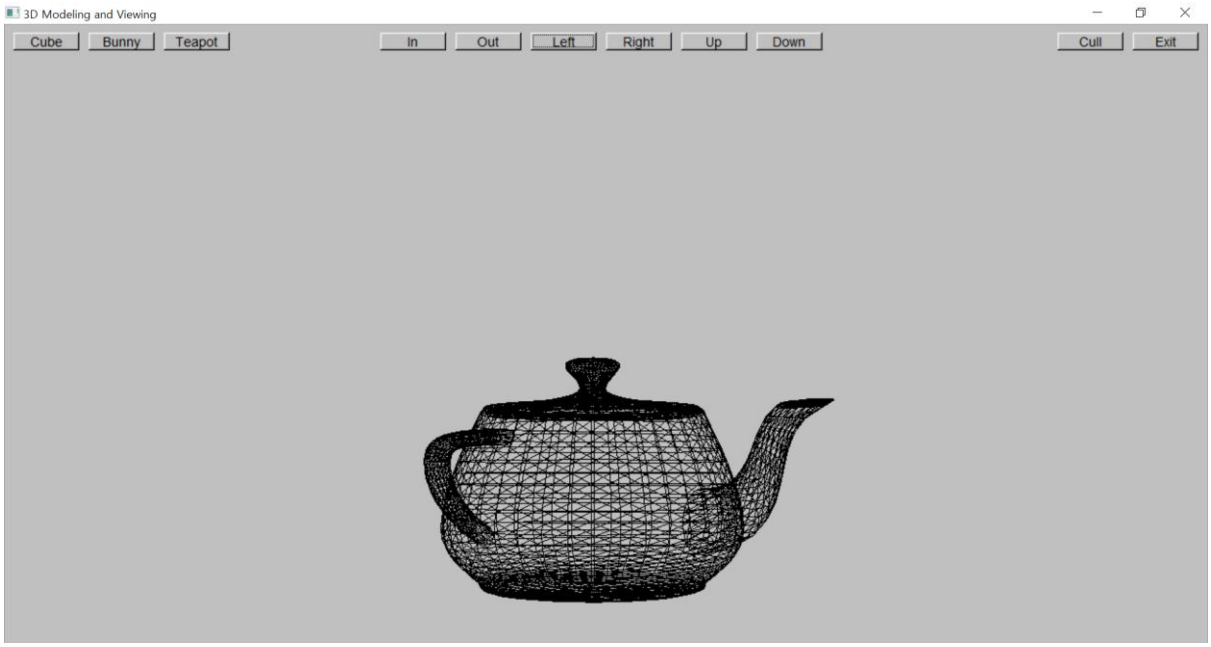
KWANGWOON UNIVERSITY

■ Begin with a summary of your results

Which requirements did you fulfill? And which didn't you?







Cube, Bunny, Teabot을 그래픽으로 구현했고 모든 버튼들이 제대로 동작된다. 그리고 Buffer와 math도 정상적으로 작동된다.

■ For each mission, explain how you fulfilled it

(중요한 기능을 가진 부분만 설명하겠습니다.)

```
VertexBuffer::VertexBuffer()
    :sz{ 0 }, space{ 0 }, buffer(nullptr)
{}

VertexBuffer::VertexBuffer(int size)
    :sz{ size }, space{ size }, buffer{ new Vertex[size] }
{
    for (int i = 0; i < sz; i++) buffer[i] = { 0,0,0 };
}

VertexBuffer::VertexBuffer(const VertexBuffer& vb)
    :sz{vb.sz}, space{vb.space}, buffer{new Vertex[vb.sz]}
{
    for (int i = 0; i < sz; ++i) buffer[i] = vb.buffer[i];
}

VertexBuffer::VertexBuffer(VertexBuffer&& vb)
    :sz{vb.sz}, space{vb.space}, buffer(vb.buffer)
{
    vb.sz = 0;
    vb.space = 0;
    vb.buffer = nullptr;
}
```

```
VertexBuffer::~VertexBuffer() { delete[] buffer; }

VertexBuffer& VertexBuffer::operator=(const VertexBuffer& vb){
    if (this == &vb) return *this;

    if (vb.sz <= space) {
        for (int i = 0; i < vb.sz; ++i) buffer[i] = vb.buffer[i];
        sz = vb.sz;
        return *this;
    }

    Vertex* p = new Vertex[vb.sz];
    for (int i = 0; i < vb.sz; ++i) p[i] = vb.buffer[i];
    delete[] buffer;
    sz = vb.sz;
    space = vb.sz;
    buffer = p;
    return *this;
}
```

```
VertexBuffer& VertexBuffer::operator=(VertexBuffer&& vb) {
    delete[] buffer;
    buffer = vb.buffer;
    sz = vb.sz;
    space = vb.space;
    vb.buffer = nullptr;
    vb.sz = 0;
    vb.space = 0;
    return *this;
}

void VertexBuffer::resize(int new_size)
{
    reserve(new_size);
    for (int i = 0; i < new_size; ++i) buffer[i]={ 0,0,0 };
    sz = new_size;
}
```

```
void VertexBuffer::reserve(int new_capacity)
{
    if (new_capacity <= space) return;
    Vertex* p = new Vertex[new_capacity];
    for (int i = 0; i < sz; ++i) p[i] = buffer[i];
    delete[] buffer;
    buffer = p;
    space = new_capacity;
}

void VertexBuffer::clear() {
    sz = 0;
}

void VertexBuffer::addVertex(float x, float y, float z)
{
    if (sz == 0)
        reserve(8);
    else if (sz == space)
        reserve(2 * space);
    buffer[sz] = { x, y, z };
    ++sz;
}
```

먼저 VertexBuffer class를 정의했다. 수업에서 배운 것처럼 메모리 관련 명령어를 사용하여 기본 생성장, 생성자, 복사 생성자, 이동 생성자, 복사 대입, 이동 대입, 소멸자를 구현했다. 그리고 메모리를 할당하는 reserve 함수를 만들고 이 함수를 통해 값을 초기화하는 resize, 값을 추가하는 addVertex 함수를 구현했다. FaceBuffer class도 이와 비슷한 방식으로 구현했고 VertexBuffer와는 다르게 Face 타입으로 메모리를 할당했다.

```
Vector3::Vector3(float x, float y, float z)
{
    // TODO: Implement here
    v[0] = x;
    v[1] = y;
    v[2] = z;
}

Vector3::Vector3(float v[3])
{
    // TODO: Implement here
    this->v[0] = v[0];
    this->v[1] = v[1];
    this->v[2] = v[2];
}
```

Vector3의 생성자는 이와 같이 구현했고 특히, float v[3]을 인자로 받는 생성자는 this를 이용하여 구현했다. Vector4의 생성자도 위와 같은 방식으로 구현했다.

```
Vector3 operator+(const Vector3& v1, const Vector3& v2)
{
    // TODO: Implement here
    return Vector3(v1.x() + v2.x(), v1.y() + v2.y(), v1.z() + v2.z());
}

Vector3 operator-(const Vector3& v1, const Vector3& v2)
{
    // TODO: Implement here
    return Vector3(v1.x() - v2.x(), v1.y() - v2.y(), v1.z() - v2.z());
}

Vector3 operator*(float k, const Vector3& v)
{
    // TODO: Implement here
    return Vector3(k * v.x(), k * v.y(), k * v.z());
}

Vector3 operator*(const Vector3& v, float k)
{
    return Vector3(k * v.x(), k * v.y(), k * v.z());
}
```

Vector3에 있는 함수를 통해 private에 접근을 했고 같은 위치에 있는 것끼리 연산을 시켰다.

Vector4의 oprator도 이와 같은 방식으로 구현했다.

```cpp
Vector3 operator^(const Vector3& v1, const Vector3& v2)
{
    // TODO: Implement here
    return Vector3(v1.y() * v2.z() - v1.z() * v2.y(), v1.z() * v2.x() - v1.x() * v2.z(), v1.x() * v2.y() - v1.y() * v2.x());
}

float operator%(const Vector3& v1, const Vector3& v2)
{
    // TODO: Implement here
    return (v1.x() * v2.x() + v1.y() * v2.y() + v1.z() * v2.z());
}
```

내적과 외적 역시 공식에 따라 같은 위치에 있는 것끼리 연산 시켰다.

```cpp
Matrix4x4 operator+(const Matrix4x4& m1, const Matrix4x4& m2)
{
    // TODO: Implement here

    return Matrix4x4(m1.operator[](0) + m2.operator[](0), m1.operator[](1) + m2.operator[](1), m1.operator[](2) + m2.operator[](2),
        m1.operator[](3) + m2.operator[](3));
}

Matrix4x4 operator-(const Matrix4x4& m1, const Matrix4x4& m2)
{
    // TODO: Implement here

    return Matrix4x4(m1.operator[](0) - m2.operator[](0), m1.operator[](1) - m2.operator[](1), m1.operator[](2) - m2.operator[](2),
        m1.operator[](3) - m2.operator[](3));

}
```

Matrix4x4의 operator는 Matrix4x4에 있는 operator[](int n)을 통해 Vector3에서 구현한 방식처럼 연산을 시켰다.

```cpp
Matrix4x4 operator*(const Matrix4x4& m1, const Matrix4x4& m2)
{
    Vector4 temp_v1[4];
    Vector4 temp_v2[4];
    Vector4 new_v2[4];
    Vector4 dot[4];

    for (int i = 0; i < 4; i++) temp_v1[i] = { m1.operator[](i)};
    for (int i = 0; i < 4; i++) temp_v2[i] = { m2.operator[](i)};

    new_v2[0] = { temp_v2[0].x(), temp_v2[1].x(), temp_v2[2].x(), temp_v2[3].x() };
    new_v2[1] = { temp_v2[0].y(), temp_v2[1].y(), temp_v2[2].y(), temp_v2[3].y() };
    new_v2[2] = { temp_v2[0].z(), temp_v2[1].z(), temp_v2[2].z(), temp_v2[3].z() };
    new_v2[3] = { temp_v2[0].w(), temp_v2[1].w(), temp_v2[2].w(), temp_v2[3].w() };

    for (int i = 0; i < 4; i++) {
        dot[i] = { temp_v1[i] % new_v2[0], temp_v1[i] % new_v2[1], temp_v1[i] % new_v2[2], temp_v1[i] % new_v2[3] };
    }

    return Matrix4x4(dot[0], dot[1], dot[2], dot[3]);
}
```

그리고 Matrix4x4의 곱은 세로의 값을 저장하는 Vector4를 이용하여 구현했다. Temp_v1 Temp_v2는 m1, m2를 저장하기 위한 배열이며 new_v2는 세로의 열을 저장하기 위해 구현했다. 그리고 한 행의 행과 열을 내적한 결과를 저장하는 dot 배열을 통해 Matrix4x4의 곱 연산자를 정의 했다.

■ Conclude with some comments on your work

Key challenges you have successfully tackled

Matrix4x4의 연산자를 정확하게 구현했기 때문에 과제를 마칠 수 있었던 것 같다.