컴퓨터 프로그래밍 C++ 프로젝트:
ASCII art로 표현하는 회전 가능한 3d 공간

강명석 (Computer Science and Engineering, 2024-10387)

이 문서는 서울대학교(SNU, Seoul National University)에서 2024년 2학기에 열린「컴퓨터 프로그래밍」강좌에서 요구하는 C++ 프로젝트 그리고 그것의 코드를 설명하기 위해 쓰였습니다.

목차

Ⅰ. Proposal

1. 개발 목표

이 프로젝트는 3d 공간에 큐브, 사각형 그리고 삼각형 등의 입체와 평면들을 생성하고, 생성한 입체를 아스키 아트로 콘솔에 출력하는 것을 목표로 합니다.
프로젝트의 소스는 github.com/KMSstudio/ascii3d 에서 볼 수 있습니다.

2. 프로젝트 사용법

사용 예시는 유튜브 영상 youtu.be/Yqwu8YJYayA에서도 확인할 수 있습니다.

A. Command window

이 창에서는 명령어를 이용하여 Space의 속성을 조회하거나, 변경할 수 있습니다. 명령어는 대소문자를 구분하지 않습니다.

Commands		
HELP	H	사용할 수 있는 명령어 리스트를 보여줍니다.
LIST	L	Camera, Screen 그리고 Object의 속성을 전부 나열합니다.
CREATE (N)	C	새로운 물체를 공간에 추가합니다. N인 인덱스 번호입니다. -1을 입력할 때 비어있는 아무 곳에 물체를 넣습니다.
ALTER [N]	A	N 번 인덱스에 있는 물체를 삭제하고 새로운 물체를 만들어 넣습니다.
DROP [N]	D	N 번 인덱스에 있는 물체를 삭제합니다.
QUIT¦SHOW	Q	현재 Command window를 나가고 화면을 Show window로 전환합니다.
EXIT¦TERM	X	프로그램을 종료합니다.

B. Show window

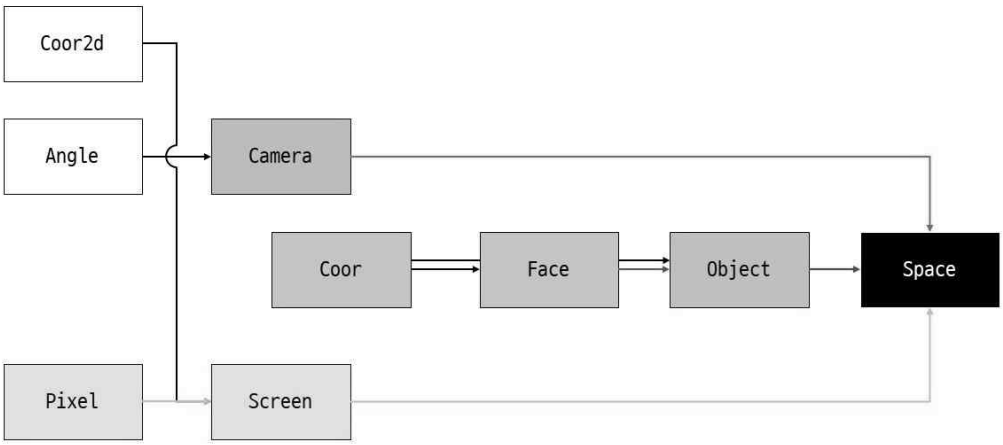
Show window 창에서는 Space를 ASCII art로 출력합니다. 공간을 회전하거나, 카메라를 이동시킬 수 있습니다. 명령어는 대소문자를 구분하지 않습니다.

Commands	
A	공간을 좌측으로 회전시킵니다.
W	공간을 위로 회전시킵니다.
S	공간을 아래로 회전시킵니다.
D	공간을 우측으로 회전시킵니다.
L	공간을 시계방향으로 회전시킵니다.
J	공간을 반시계 방향으로 회전시킵니다.
I	카메라를 더 가까이 가져갑니다.
J	카메라를 더 멀리 가져갑니다.
Q	Show window를 종료하고 Command window를 엽니다.
X	프로그램을 종료합니다.

Ⅱ Objects

소유관계

A → B는 B가 A를 매개변수로써 소유하고 있음을 나타냅니다.



객체 리스트

Objects		
Angle	3차원상의 라디안 각	base.hpp
Coor2d	2차원상의 정수 점	base.hpp
Pixel	스크린의 한 픽셀. 정수점에 대응	screen.hpp
Screen	스크린	screen.hpp
Camera	카메라	camera.hpp
Coor	3차원상의 점	face.hpp
Face	면	face.hpp
Body	물체	body.hpp
Space	공간이자 컨트롤러	space.hpp

Ⅲ. Design Considerations

Face::clone()의 존재 이유

Face 객체는 사실 객체 하나만 놓고 보면 매개변수가 전부 원시 타입이므로, `Face* fp = f.clone();`가 더 간단한 문법 `Face* fp = new Face(f);`로 대체될 수 있을 것처럼 보인다. 그러나 이 부분은 `Body` 클래스가 `Face` 객체의 다형성을 보존하며 DeepCopy 되기 위해 존재하는 것이므로 반드시 구현되어 있어야 한다.

근본적으로 `Space` 객체는 `Space::show()`에서 자신이 가지고 있는 `std::vector<Body*> body;`를 `std::vector<Body*> showBody;`로 옮겨와야 한다. 이때 `Body*`값을 그대로 가져와서는 안 되므로 `Body`를 DeepCopy한다.

DeepCopy될 `Body`는 `std::vector<Face*> face;`를 매개변수로 가지고 있다. 마찬가지로 `Face*`값을 그대로 사용할 수는 없으므로 `Face`를 DeepCopy하게 된다.

이때, `Face*`는 다형성을 가진다. 실질적으로 face 벡터 안에는 `Face`를 상속한 `Square`, `Triangle`의 주소가 있는 것이다. 따라서 DeepCopy를 `push_back(new Face(*f));`처럼 진행하면 다형성이 깨지게 되므로 해서는 안 된다.[1)]

이 문제를 해결하기 위해 프로그램은 `Face::clone()` 가상함수를 만들어 사용하게 된다. `Face`를 상속받는 객체에서도 적절히 clone 함수를 만든 뒤, `push_back(f->clone());`과 같이 사용해 다형성을 보존한다.

객체 Coor의 존재 위치

객체 Coor는 3d 표현에 있어 매우 기본적인 입자인데, base.hpp가 아닌 face.hpp에 작성한다. 이것은 메소드 `Coor::project`가 `Screen`과 `Camera`를 매개변수로 받기 때문이다. screen.hpp와 camera.hpp가 base.hpp를 참조하고 있으므로, `Coor`을 base.hpp에 기술하면 순환 참조가 되어 사용하지 못한다.

객에 Space와 project 메소드에서 unit의 존재 의의

unit은 삼차원 공간인 space와 이차원 공간인 screen의 변환에 사용되는 상수다. unit의 정의는 다음과 같다.

> "Space를 가득 채우는 어떤 Cube가 있다고 하자. 이 공간은 Camera에 의해 Screen에 투영된다. 이때 투영된 상이 전체 Screen 면적의 p(0≤p≤1)만큼을 차지한다."

식으로는 $\text{unit} = p\sqrt{\frac{x_c y_c}{x_s y_x}} \cdot (z_c + \text{depth})$로 나타낼 수 있다. x_c는 공간(Cube)의 x크기, x_s는 스크린(Screen)의 x크기이고, depth는 카메라의 초기 깊이다. 프로그램에서는 $p = 0.5$로 설정하였다.

---

1) 정확히는 다형성이 깨지면서 override 했던 `Square::project`, `Triangle::project`가 전부 `Face::project`로 되어 버리기 때문.

객체 Square::project에서 fillGap의 존재 의의

`fillGap`은 비어있지 않은 스크린 출력을 위해 존재한다. 값의 의미는 다음과 같다.

> "삼차원 공간에서 거리 차가 `fillGap` 이하인 두 점 A, B가 있을 때, 두 점이 스크린에 표시되는 위치 차는 1픽셀 이하임이 보증된다."

선분 (0, 0)~(1, 0)을 출력하려고 할 때, 점들을 어떤 간격으로 출력할지에 대한 문제가 있었다. 간격이 너무 촘촘하면 속도가 느려지고, 간격이 너무 넓으면 픽셀이 비어버린다. 이때 간격을 `fillGap`으로 출력하면 중간에 비어있는 픽셀 없이 모든 픽셀을 스크린에 표시할 수 있다. 식으로 아래처럼 계산한다.

$$\text{fillGap} = \frac{\min(z) + \text{depth}}{\text{unit}}$$

이때 depth는 unit과 달리 현재 카메라의 깊이다. 프로그램에서는 혹시 모를 누락을 위해 이 값에 `SQ_FILL_GAP = 0.5`를 곱하여 실제 `fillGap`으로 사용한다.

Space에서 물체의 회전

객체 `Spcae`에서는 물체들을 회전시킬 때 카메라를 이동시키는 방법을 사용하지 않는다. 대신 객체 `Space`는 물체 전체를 `std::vecotr<Body*> showBody`로 복사한 뒤, 복사한 물체들을 회전시킨다. 이런 회전방법을 사용하는 이유는 만약 물체가 아니라 카메라를 이동시킬 시 w/a/s/d를 통한 직관적 이동에 문제가 생길 수 있기 때문이다.

예를 들어, 카메라를 w키로 $+\theta\,\pi/2$만큼 이동시킨 다음 명령 a를 수행하면 카메라는 좌편으로 이동하는 게 아니라 $+y$축을 중심으로 회전하는 것처럼 보인다. 이런 일을 방지하기 위해 좌표계 전체를 이동시키는 방안을 채택하였다.

Ⅳ. Parameters & Methods

객체 Angle (base.hpp)

공간의 회전을 지칭합니다. 회전 theta, phi를 정의합니다. 각의 단위는 라디안입니다.

Parameters	
`float p;`	phi각 φ입니다. public 값입니다.
`float t;`	theta각 θ입니다. public 값입니다.
`float z;`	z축 방향 회전각입니다. public 값입니다.
Methodes	
`Angle();`	임의의 `Angle`을 생성합니다. 기본 좌표는 p=t=0.
`Angle(float p, float t);`	주어진 좌표를 가지는 `Angle`을 생성합니다.

객체 Coor2d (base.hpp)

2d 좌표 객체입니다. Screen과 pixel이 활용할 것이기 때문에, x와 y 값은 정수입니다.

Parameters	
`int x;`	x 좌표입니다. public 값입니다.
`int y;`	y 좌표입니다. public 값입니다.
Methodes	
`Coor2d();`	임의의 `Coor2d`를 생성합니다. 기본 좌표는 x=y=0.
`Coor2d(int x, int y);`	주어진 좌표를 가지는 `Coor2d`를 생성합니다.

객체 Pixel (screen.hpp)

`Pixel` 객체는 카메라가 촬영한 한 픽셀을 지칭합니다. 각 픽셀은 깊이 depth, 출력할 문자열 ch등을 가집니다. 출력할 문자열을 ch가 아니라 depth에 따라 달라지는 문자열로 설정할 수도 있습니다.

Parameters	
`char ch;`	픽셀의 값입니다.
`float depth;`	인식된 픽셀의 깊이입니다. z값을 뜻합니다.
`bool empty;`	픽셀이 비어있는지 확인하는 값입니다.
Methodes	
`Pixel();`	ch의 값은 ' '로, depth의 값은 -1로 지정합니다.
`Pixel(char ch, float depth);`	ch 값과 depth의 값을 지정합니다.
`char get() const;`	ch 값을 반환합니다.
`int set(const Pixel& pixel);`	만약 입력받은 pixel.depth < this.depth 이거나 this.empty==1이라면, ch, depth를 pixel의 것으로 바꿉니다. empty는 0이 됩니다.
`int set(char ch, float depth);`	
`void clear();`	

객체 Screen (screen.hpp)

Screen 객체는 카메라가 촬영하는 스크린을 지칭합니다.

Parameters	
std::vector<Pixel> pixels;	각 픽셀을 지정합니다.
Coor2d size;	스크린의 크기입니다. 크기는 원점에서 this->size까지입니다. this->size를 불포함합니다.
Coor2d center;	스크린의 중앙입니다.
Methodes	
Screen(Coor2d size);	size를 지정하고 this->size.x * this->size.y개의 Pixel을 할당합니다.
~Screen();	Destructor입니다. std::vector를 사용했기 때문에, 별도의 메모리 해제 작업을 거치지 않습니다.
Pixel getPixel(const Coor2d& pos) const;	위치 pos의 픽셀을 가져옵니다.
char prtPixel(const Coor2d& pos) const;	위치 pos에 있는 픽셀의 출력 문자를 가져옵니다. Pixel::get을 시행합니다.
void setPixel(const Coor2d& pos, const Pixel& pixel);	위치 pos의 픽셀을 지정합니다. Pixel::set을 사용합니다.
void clsPixel(const Coor2d& pos);	위치 pos의 픽셀을 삭제합니다. Pixel::clear을 사용합니다.
Coor2d getCenter() const;	중심위치를 반환합니다.
Coor2d getSize() const;	크기를 반환합니다.
void clear();	스크린 전체를 비웁니다.
int print(void) const;	스크린 전체를 출력합니다. 모든 픽셀에 대해 Pixel::get을 시행한 후, 가져온 문자를 출력합니다.
std::string prtExp() const;	printExport, 스크린 전체를 문자열로써 내보냅니다.

객체 Camera (camera.hpp)

Camera 객체는 공간을 촬영하는 카메라를 지칭합니다.

Parameters	
Angle angle;	카메라의 회전각도입니다. Public입니다.
float depth;	카메라와 Space::center 간의 거리입니다. Public.
float depthMin;	가능한 최소거리입니다.
float depthMax;	가능한 최대거리입니다.
Methodes	
Camera(float depth = 100, float depthMin = 20, float depthMax = 200);	
Camera를 생성합니다. 회전각을 (0, 0)으로, 나머지는 인자값으로 설정합니다.	
int act(char move);	

	카메라를 움직입니다. 가능한 움직임은 6가지가 있습니다. (a/w/s/d/i/j) 그러나 실제로 사용하는 움직임은 i와 j뿐입니다. 움직임에 대한 설명은 Ⅰ.2.B 참고.
`void reset(float depth = 100);`	카메라의 위치를 초기화합니다. 회전각을 (0, 0)으로 만들고, depth를 설정합니다.
`void setDepthMin(float depthMin);`	
`float getDepthMin() const;`	
`void setDepthMax(float depthMax);`	
`float getDepthMax() const;`	
`Angle getAngle() const;`	
`float getDepth() const;`	

객체 Coor (face.hpp)

공간에 존재하는 한 좌표를 지칭합니다.

Parameters	
`float x, y, z;`	3차원 좌표입니다. public값입니다.
Methodes	
`Coor();`	임의의 `Coor`를 생성합니다. 기본 좌표는 x=y=z=0입니다.
`Coor(float x, float y, float z);`	주어진 좌표를 가지는 `Coor`를 생성합니다.
`float abs(void) const;`	벡터의 크기를 반환합니다.
`Coor unit(void) const;`	방향이 같은 단위벡터를 생성합니다.
`Coor rotate(const Angle angle) const;`	(0, 0, 0)을 중심으로 angle만큼 회전시킵니다.
`Coor rotate(const Coor& center, const Angle angle) const;` center를 중심으로 anlge만큼 회전시킵니다.	
`int position(const Screen& screen, const float cameraDepth, const float unit, Coor2d& pos) const;` 주어진 screen와 cameraDepth에 대해, 좌표가 표시되는 screen의 픽셀 위치를 계산해 pos에 저장합니다. 좌표가 카메라에 잡히지 않는다면 -1을 반환합니다.	
`int project(const Camera& camera, const char ch, const float unit, Screen& screen);` 주어진 screen, camera에 대해 해당 좌표의 픽셀 위치를 계산한 다음 screen을 업데이트합니다. 이때 문자는 ch입니다. unit은 2d과 3d간 변환에 사용되는 비로, `Space`에서 계산한 값입니다.	
Operators	
`Coor operator+(const Coor& other) const;`	두 좌표를 더합니다.
`Coor operator-(const Coor& other) const;`	두 좌표를 뺍니다.
`Coor operator*(float scalar) const;`	좌표에 실수를 곱합니다.
`Coor operator/(float scalar) const;`	좌표에 실수는 나눕니다.

`Coor operator-() const;`	좌표에 -1을 곱합니다.

객체 Face (face.hpp)

인터페이스 `Face`는 기본적인 면을 구성합니다. 이후 `Face`를 상속하는 객체가 project 메소드와 coorSize를 정의해야 합니다.

Parameters	
`char ch;`	면의 출력문자입니다.
`std::vector<Coor> coor;`	`Coor`의 집합입니다. 면을 정의하는 점들입니다.
Methodes	
`Face(char ch, int coorSize);`	`Face`를 생성합니다.
`~Face() = default;`	
`virtual Face* clone() const = 0;`	자신과 같은 객체를 생성한 후 주소를 반환합니다. `Body`의 DeepCopy를 위해 사용됩니다.
`void rotate(const Angle angle);` 모든 점을 (0, 0, 0)을 중심으로 anlge만큼 회전시킵니다.	
`void rotate(const Coor& center, const Angle angle);` 모든 점을 center를 중심으로 anlge만큼 회전시킵니다.	
`virtual int project(const Camera& camera, const float unit, Screen& screen) const;` 모든 점을 project합니다. unit은 `Space`에서 계산된 값입니다.	

객체 something:Face (face.hpp)

Objects	
`Square`	사각형입니다. 한 점과 두 벡터로 정의됩니다. 사각형은 $s + (av_1 + bv_2)$ ($0 \leq a, b \leq 1$) 입니다.
`Triangle`	삼각형입니다. 한 점과 두 벡터로 정의됩니다. 삼각형은 $s + (av_1 + bv_2)$ ($0 \leq a, a + b \leq 1$) 입니다.

객체 Body (body.hpp)

인터페이스 `Body`는 기본적인 물체를 구성합니다. 이후 `Body`를 상속하는 객체가 face의 크기와 내용을 정의해야 합니다.

Parameters	
`std::vector<Face*> face;`	면의 집합입니다. 다형성(polymorphism)을 지원받기 위해 면을 포인터 형태로 저장합니다.
`Coor coor[2];`	`Body`의 크기입니다. coor[0]-coor[1]까지의 공간을 의미합니다.
`Coor center;`	`Body`의 중앙입니다. config로 물체를 회전시킬 경우, 이

점을 기준으로 회전합니다.
Methodes
`void rotate(const Angle angle, const int byCenter = 0);` 모든 점을 (0, 0, 0)을 중심으로 anlge만큼 회전시킵니다. 만약 byCenter가 1이라면 this->center를 중심으로 회전시킵니다.
`void rotate(const Coor& center, const Angle angle);` 모든 점을 center를 중심으로 angle만큼 회전시킵니다.
`virtual int project(const Camera& camera, const float unit, Screen& screen);` 모든 면을 project합니다. unit은 Space에서 계산된 값입니다.

객체 something:Body (body.hpp)

Objects	
`Cube`	직육면체입니다. 한 점과 세 벡터로 정의됩니다. 직육면체는 $s + (av_1 + bv_2 + cv_3)$ ($0 \leq a, b, c \leq 1$) 입니다.
`Facebody`	면입니다. 생성될 때 `Face*`를 받고 이것을 DeeoCopy해 사용합니다. 면 하나를 Body처럼 사용하기 위해 만들어졌습니다.

객체 Space (space.hpp)

Space 객체는 공간 전체를 지칭합니다.

Parameters	
`Body* body[16];`	공간에 존재하는 body의 리스트입니다. 기본상태는 0x0입니다.
`Body* showBody[16];`	Space::show();에서 사용하는 body의 복사본입니다.
`Camera* camera;`	카메라입니다.
`Screen* screen;`	스크린입니다. 모든 body가 투영됩니다.
`Coor size;`	Space의 크기입니다. 공간은 -size에서 size까지의 크기를 가집니다.
`float unit;`	screen에 투영을 진행할 때 사용되는 상수입니다. camera와 screen 값이 모두 있다면, 자동으로 계산됩니다.
Methodes	
`Space(const Coor& size);`	
임의의 `Space`를 생성합니다. 모든 포인터값은 `0x0`입니다.	
`~Space();`	
`void setScreen(const Coor2d& screenSize);`	
새로운 Screen 객체를 만들어 할당합니다.	

`void setCamera(float depth, float minDepth, float maxDepth);`	
새로운 Camera 객체를 만들어 할당합니다.	
`int _calc();`	screen와 camera가 있을 때 unit을 계산합니다.
`void _move(const char move);`	showBody를 입력에 따라 회전시킵니다. Space::show()에서 호출합니다.
`void _config(const char config);`	입력에 따라 객체의 속성을 출력하거나 변경합니다. Space::config()에서 호출합니다.
`void Space::_make(int index=-2);`	새로운 물체를 생성합니다. `std::cout`, `std::cin`을 사용하여 사용자에게 직접 입력을 받습니다. Space::_config()에서 호출합니다.
`int show(int verbose = 0);`	
보유한 body를 showBody로 로드해 출력합니다. verbose가 1이라면 간략한 정보를 함께 표시합니다. 반환값 0은 함수 종료, -1은 프로그램 종료를 의미합니다.	
`int config();`	
계속 입력을 받고 그에 맞춰 _config를 호출합니다. 반환값 0은 함수 종료, -1은 프로그램 종료를 의미합니다.	
`int make(const Body& newBody, int index = -1);`	
새로운 Body를 body에 추가합니다. 0부터 15 사이의 index를 지정할 수 있고, index = -1일 경우 비어있는 곳에 Body를 추가합니다.	

Ⅴ. Future Enhancements

각 Body의 특정 점 편집

AutoCAD 또는 Fusion360과 같은 프로그램의 경우, Cube에서 한 정점만 잡아 늘인다거나, 한 정점만 삭제할 수 있는 기능을 지원하고 있다. 이 프로그램에서도 이러한 기능을 지원하게 개선할 수 있을 것이다. 각 Body의 Face를 전부 Triangle로 만들거나, Body에 Face 대신 Coor을 만드는 방향으로 구현할 수 있을 것으로 예상한다.

거리에 따라 더 어두워지는 Pixel

현재 Screen의 출력은 z 거리와 무관하게 ch 값을 출력하는 방향으로 구현돼 있다. 이때, 입체감을 위해 출력값이 z 거리에 따라 달라지도록 개선하는 것을 고려할 수 있다. 픽셀이 표시하는 공간과 카메라의 거리에 따라 ch를 더 밝기가 낮은 ch로 만들거나 (‘@’ → ‘#’), 콘솔에서 출력하는 문자의 색상을 변경하는 방식으로 (7 → f → 8 → 0) 구현할 수 있을 것이다.

Space의 파일 출력 기능

마찬가지로 AutoCAD 또는 Fusion360과 같은 프로그램의 경우, 만든 프로젝트를 파일로 저장한 다음 다시 불러올 수 있다. 이처럼, 본 프로그램에서도 Space가 가진 body들을 내보내고, 불러오는 기능을 추가할 수 있다. 클래스의 매개변수들을 BIN 파일로 그대로 저장하는 방법, Space에 매개변수도 있는 `Body`들과 `Camera`, `Screen`의 정보를 텍스트로 변환해서 저장하는 방법을 고려할 수 있다.

다양한 Body 객체 지원

현재 Space 공간에 추가할 수 있는 객체는 `Cube`와 `Facebody`이다. 사용할 수 있는 객체가 적기에, Cone, cylinder 또는 Arch 등 다른 `Body` 객체를 추가하는 방향으로 프로그램을 개선할 수 있다. 단순히 `Body` 객체를 상속하는 객체를 만듦으로써 구현할 수 있으므로, 구현 또한 간단할 것으로 예상된다.

다양한 Face 객체 지원

현재 사용할 수 있는 `Face` 객체는 `Square`과 `Triangle`뿐이다. 사실 일반적으로 `Body` 객체가 활용할 `Face`는 `Triangle`과 `Square`가 거의 전부겠지만, 사용자가 직접 `Face`를 `Space`에서 추가할 수 있다는 점, 그리고 cylinder같이 `Triangle`만으로 표현하기에 복잡할 수 있는 물체를 구현할 계획이 있다는 점을 고려할 때, oval, hexagon과 같은 객체를 추가로 구현할 수 있다.

Ⅵ. Note

Date	Task
24-09-14	프로젝트 목적과 필요한 객체의 목록 작성
24-09-15	구체적인 객체의 관계 작성 base, screen, camera 작성
24-09-16	camera.angle과 camera.depth를 public으로 변경하고 getAngle, getDepth 메소드 추가 screen에 getCenter, getSize 매소드 추가 face.hpp 작성
24-09-16	face.hpp 수정 및 기작성 클래스 구조개선 body.hpp 작성 screen.hpp 작성
24-09-17	space.hpp 작성 ascii3d.hwpx 문서 작성 버그 수정
24-09-18~23	버그 수정 Space::_make 작성

Reference

[1] servetgulnaroglu, “cube.c,” GitHub, github.com/servetgulnaroglu/cube.c, commit 7039318, Accessed: Sep. 16, 2024.
[2] 김홍종, 미적분학, SNU Press, 2023.