メディアプログラミング演習

第8回参考資料

ヒント

これはあくまでヒントなので内容を写しても動くとは限りません

カメラのオブジェクト

- ofApp.h の ofApp クラスに以下の宣言を追加する
  - 例）`ofCamera camera;`
    - もしくは
  - 例）`ofEasyCam camera;`

ライトのオブジェクト

- ofApp.h の ofApp クラスに以下の宣言を追加する
  - 例）`ofLight light;`

砲弾の速度と加速度

- ofApp.h の ofApp クラスの宣言より前に以下を追加する
  - `using namespace glm;`
  - これを入れないときは以降の `vec3` を全部 `glm::vec3` にする
- ofApp.h の ofApp クラスに以下の宣言を追加する
  - 例）`vec3 velocity, gravity;`

図形のオブジェクト

- ofApp.h の ofApp クラスに使用する図形のクラスのオブジェクトの宣言を追加する
  - 回転台 ofCylinderPrimitive クラス（円柱）
    - 例）`ofCylinderPrimitive turn_table;`
  - 台座 ofBoxPrimitive（直方体）
    - 例）`ofBoxPrimitive left_pedestal, right_pedestal;`
  - 砲身 ofCylinderPrimitive クラス（円柱）
    - 例）`ofCylinderPrimitive barrel;`
  - 砲弾 ofSpherePrimitive クラス（四角形で構成された球）
    - 例）`ofSpherePrimitive bullet;`
  - 地面 ofPlanePrimitive（平面）
    - 例）`ofPlanePrimitive ground;`

カメラの位置、方向、画角の設定

- カメラの位置
  - void ofNode::setPosition(const vec3 &p)
    - 例）`camera.setPosition(vec3{ 0.0f, 10.0f, 50.0f });`
- カメラの方向
  - ofNode::lookAt(const glm::vec3 &t)
    - 例）`camera.lookAt(vec3{ 0.0f, 10.0f, 0.0f });`
- カメラの画角（単位は度）
  - ofNode::setFov(float f)
    - 例）`camera.setFov(30.0f);`

ライトの位置／方向の設定と有効化

- ライトの位置（点光源の場合）
  - void ofNode::setPosition(const glm::vec3 &p)
    - 例）`light.setPosition(vec3{ 100.0f, 300.0f, 200.0f });`
- ライトの方向（平行光線の場合）
  - ofNode::lookAt(const glm::vec3 &t)
    - 例）`light.lookAt(vec3{ 0.0f, 10.0f, 0.0f });`
- ライトの有効化
  - void ofLight::enable()
    - 例）`light.enable();`

【参考】点光源と平行光線

- 光源を点光源にする（デフォルト）
  - void ofLight::setPointLight()
    - 例）`light.setPointLight();`
- 光源を平行光線にする
  - void ofLight::setDirectional()
    - 例）`light.setDirectional();`
- 点光源は位置の変化が陰影に影響するが向きは影響しない
- 平行光線は向きの変化が陰影に影響するが位置は影響しない

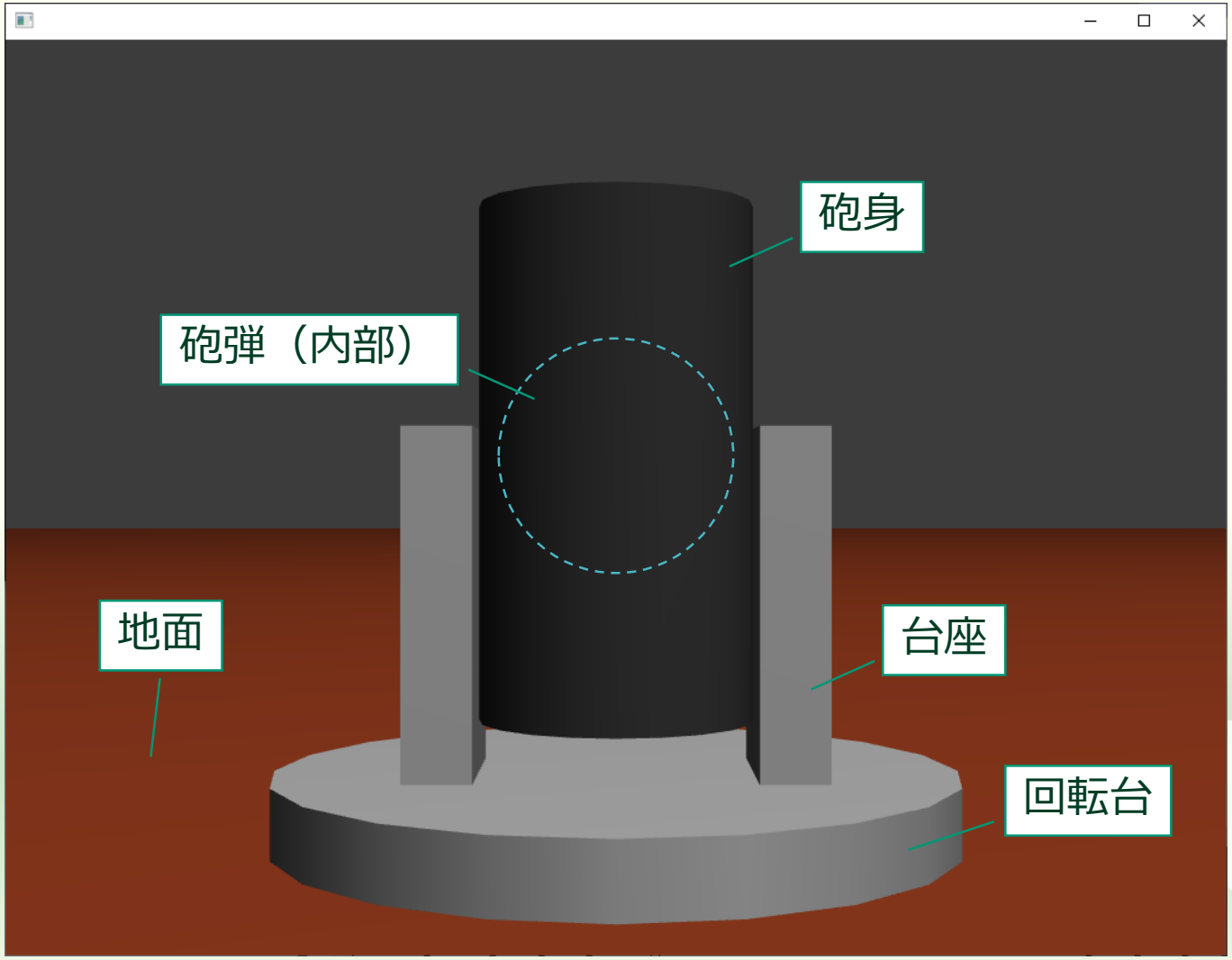
隠面消去処理の有効化

- デプステストを有効にすると隠面消去処理が行われる
  - void ofEnableDepthTest()
    - 例）`ofEnableDepthTest();`

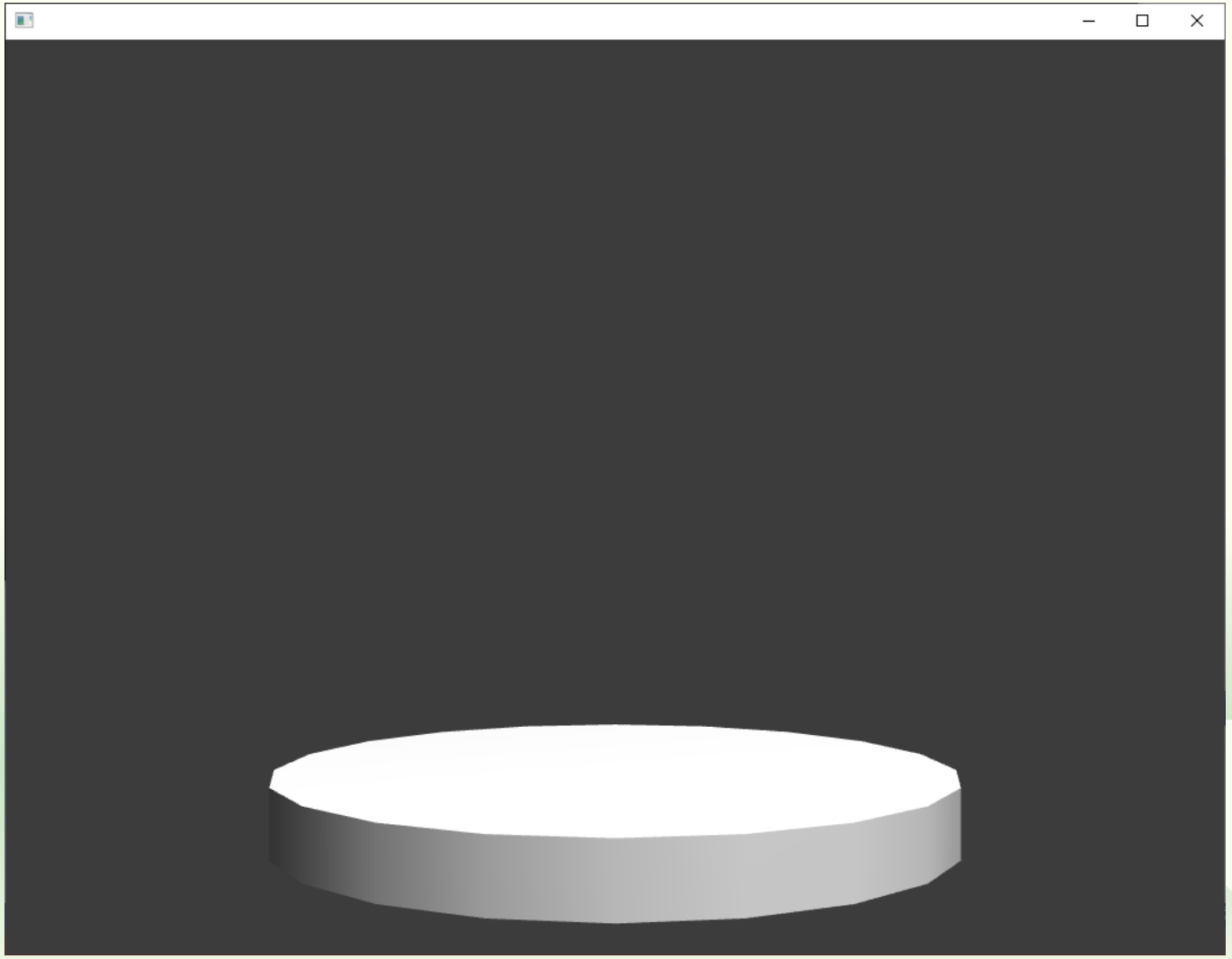
砲弾の速度と加速度の初期値の設定

- 砲弾は最初に砲身の内部で静止している
  - 速度も加速度も０にする
    - 例）`velocity = gravity = vec3{ 0.0f, 0.0f, 0.0f };`

図形のオブジェクトの設定

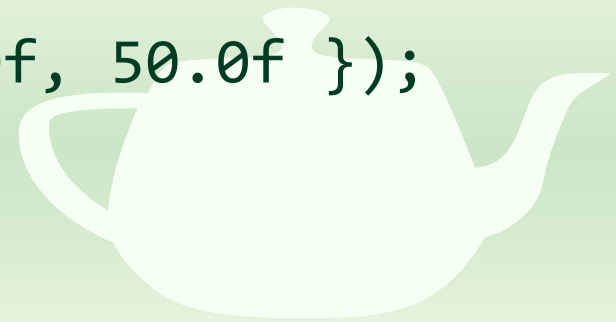


回転台の作成

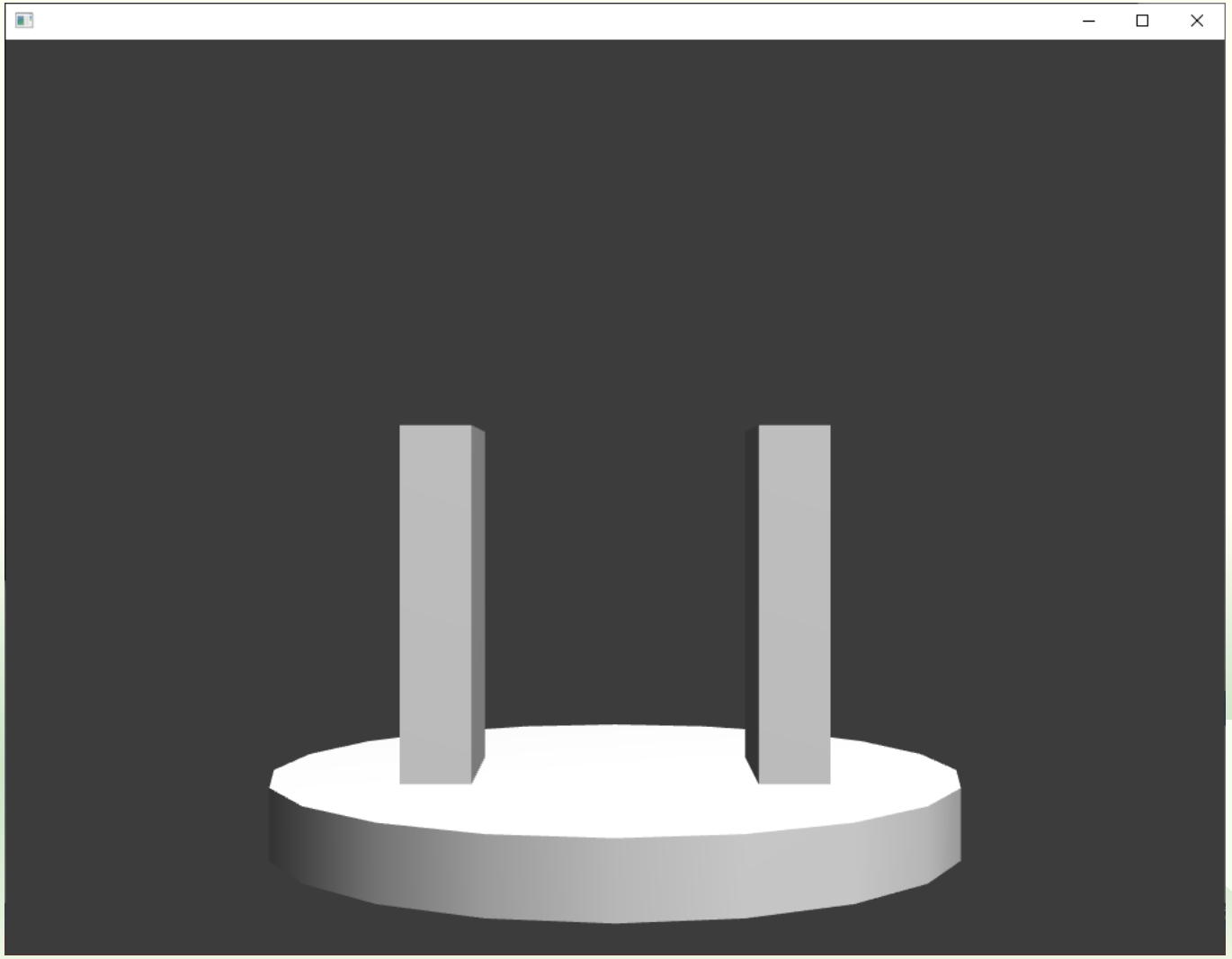


回転台の設定

- 大きさと解像度
  - void ofCylinderPrimitive::set(float radius, float height, int radiusSegments, int heightSegments, int capSegments=2, bool bCapped=true, ofPrimitiveMode mode=OF_PRIMITIVE_TRIANGLE_STRIP)
    - 例）`turn_table.set(10.0f, 2.0f, 40, 1);`
- 位置
  - void ofNode::setPosition(const vec3 &p)
    - 例）`turn_table.setPosition(vec3{ 0.0f, 10.0f, 50.0f });`

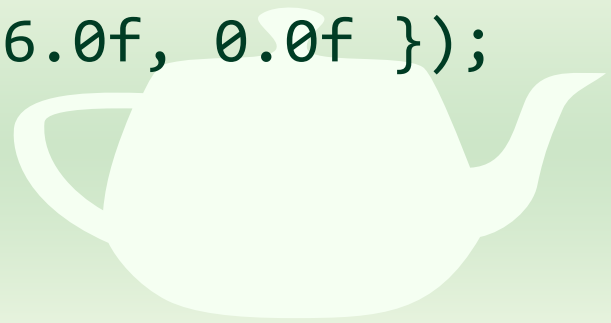


回転台に台座を追加



台座の設定

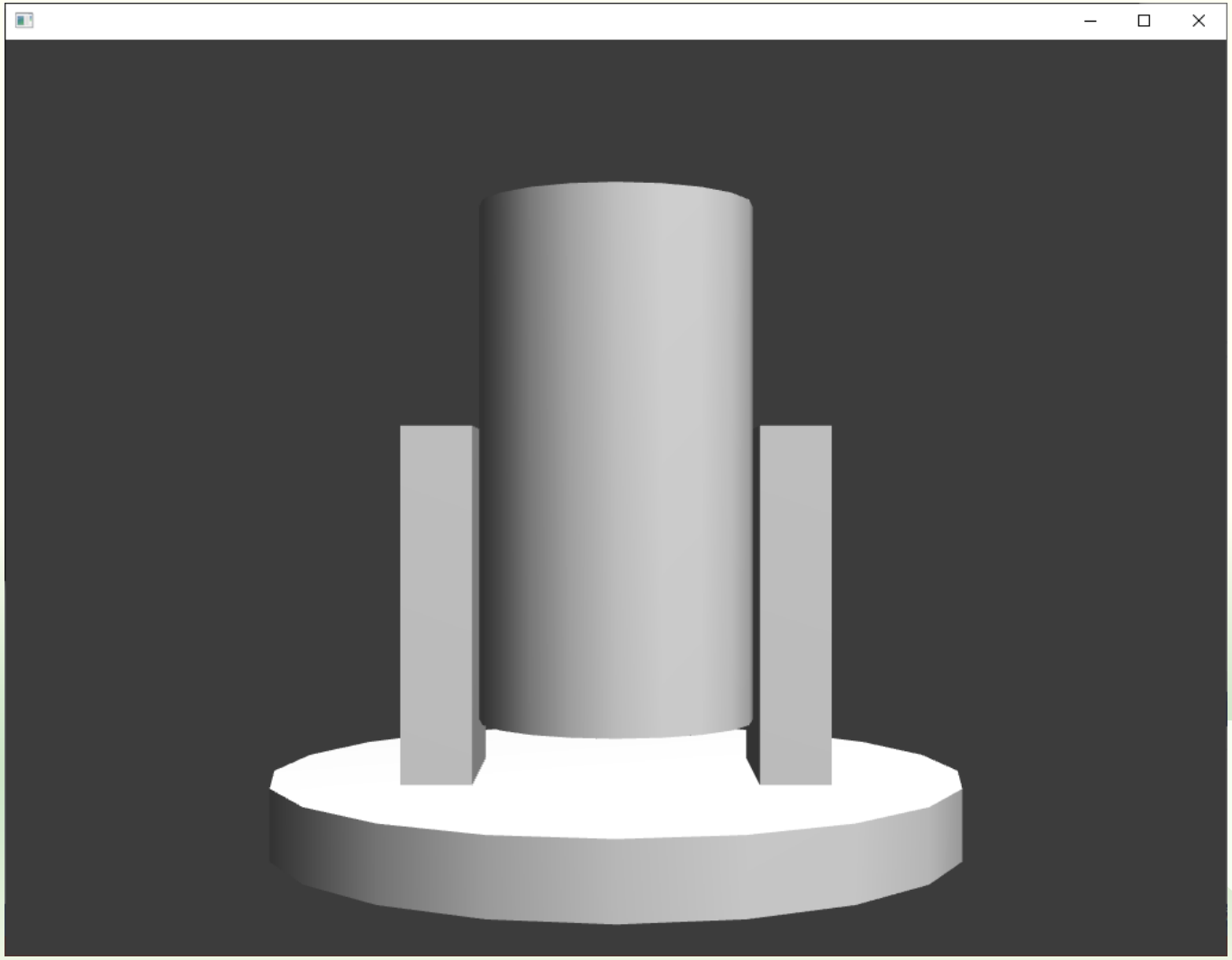
- 大きさ
  - void ofBoxPrimitive::set(float width, float height, float depth)
    - 例）
```
left_pedestal.set(2.0f, 10.0f, 5.0f);
right_pedestal.set(2.0f, 10.0f, 5.0f);
```
- 位置
  - void ofNode::setPosition(const vec3 &p)
    - 例）
```
left_pedestal.setPosition(vec3{ -5.0f, 6.0f, 0.0f });
right_pedestal.setPosition(vec3{ 5.0f, 6.0f, 0.0f });
```



親子関係

- 台座の親を回転台にする
  - void ofNode::setParent(ofNode &parent, bool bMaintainGlobalTransform=false)
    - 例）
```
left_pedestal.setParent(turn_table);
right_pedestal.setParent(turn_table);
```

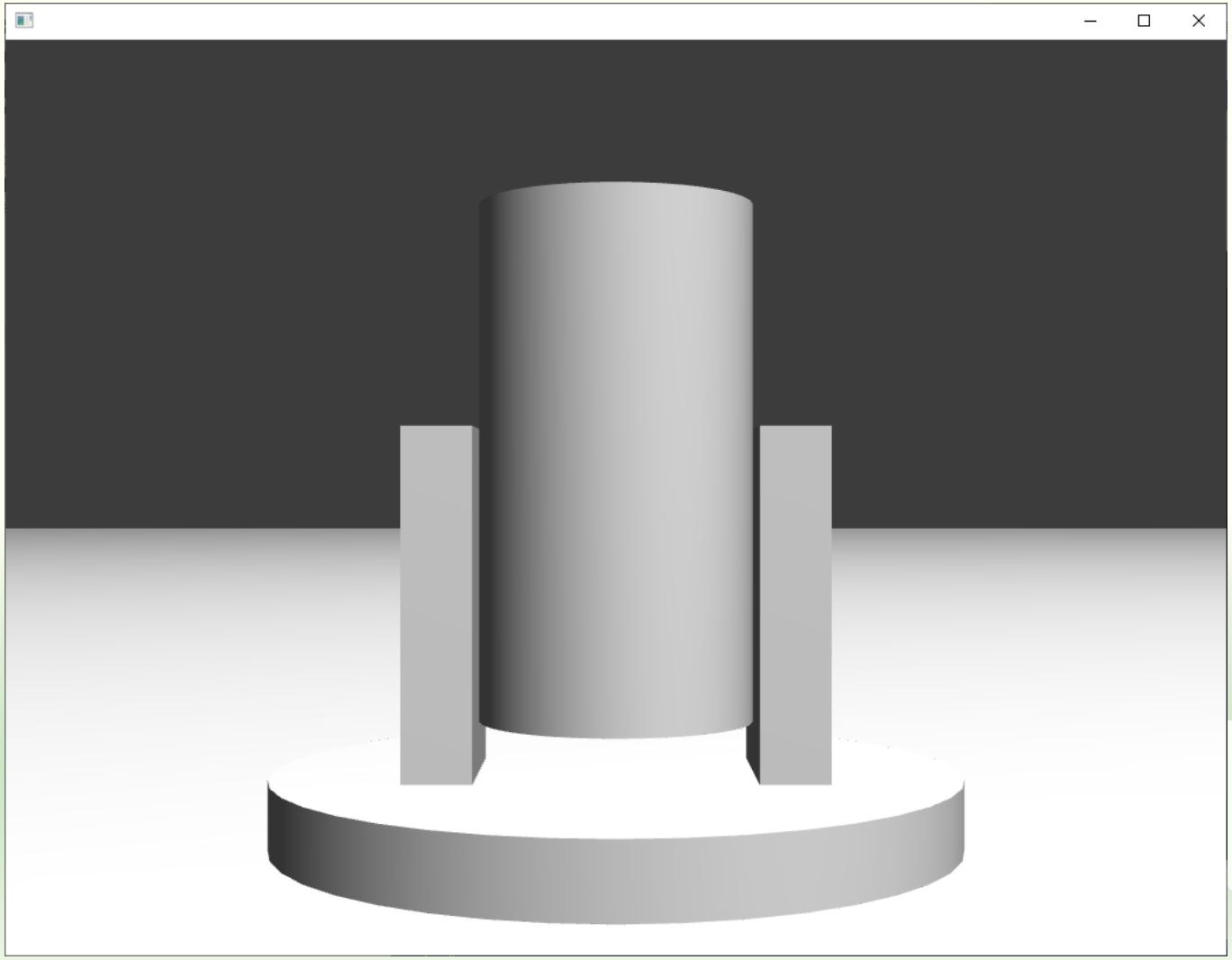
回転台に砲身を追加



砲身の設定

- 大きさ
  - void ofCylinderPrimitive::set(float radius, float height, int radiusSegments, int heightSegments, int capSegments=2, bool bCapped=true, ofPrimitiveMode mode=OF_PRIMITIVE_TRIANGLE_STRIP)
    - 例）`barrel.set(4.0f, 15.0f, 40, 1);`
- 位置
  - void ofNode::setPosition(const vec3 &p)
    - 例）`barrel.setPosition(vec3{ 0.0f, 10.0f, 0.0f });`
- 回転台を親にする
  - void ofNode::setParent(ofNode &parent, bool bMaintainGlobalTransform=false)
    - 例） `barrel.setParent(turn_table);`

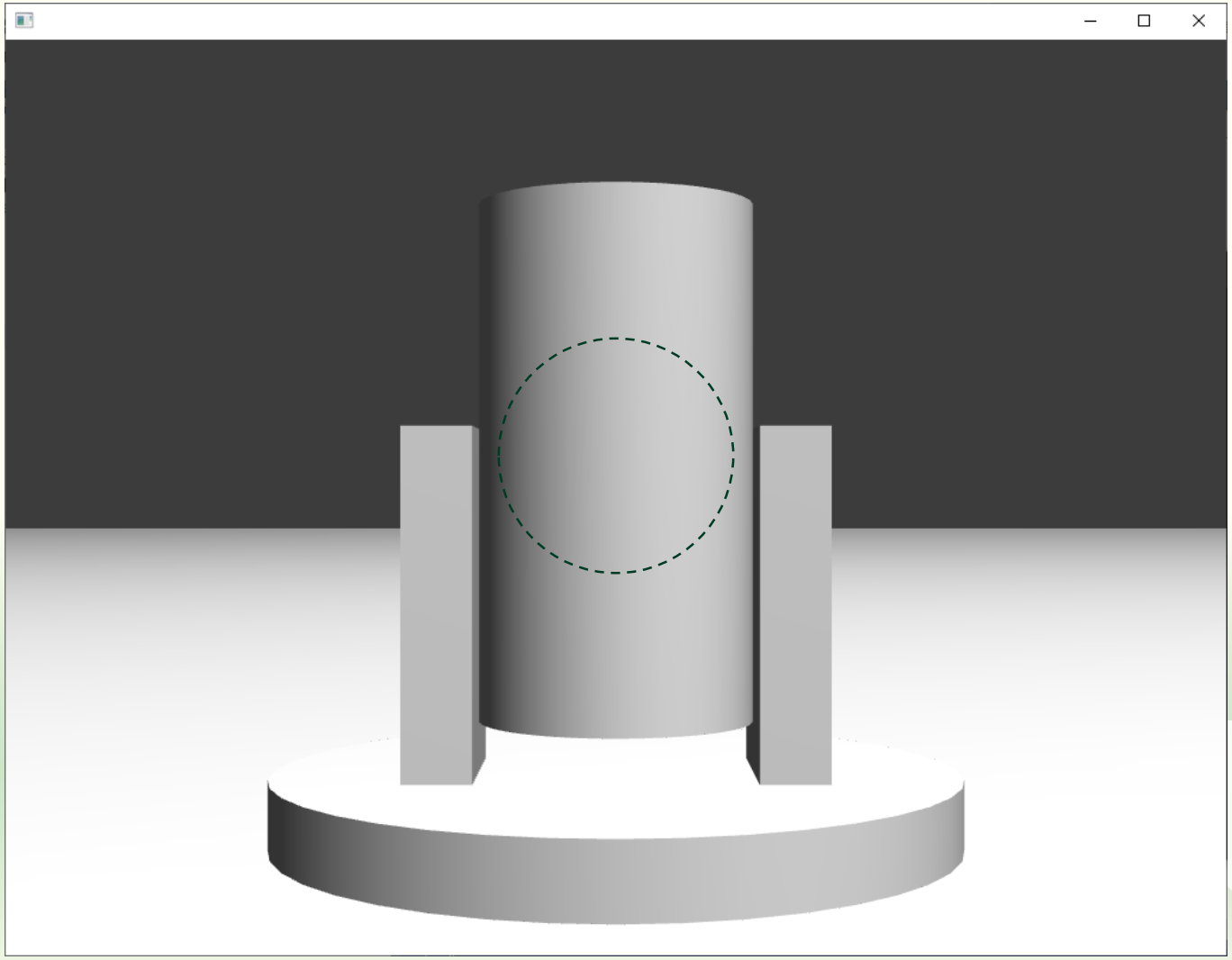
地面を追加



地面の設定

- 大きさと解像度
  - void ofPlanePrimitive::set(float width, float height, int columns, int rows, ofPrimitiveMode mode=OF_PRIMITIVE_TRIANGLE_STRIP)
    - 例）`plane.set(1000.0f, 1000.0f, 100, 100);`
    - 解像度を上げておかないときれいな陰影が出ない
- x軸中心に-90度回転して上に向ける
  - void ofNode::tiltDeg(float degrees)
    - 例）`ground.tiltDeg(-90.0f);`

砲弾を追加



回転台の設定

- 大きさと解像度
  - void ofSpherePrimitive::set(float radius, int resolution, ofPrimitiveMode mode=OF_PRIMITIVE_TRIANGLE_STRIP)
    - 例）`bullet.set(3.0f, 20);`
- 位置
  - void ofNode::setPosition(const vec3 &p)
    - 例）`bullet.setPosition(0.0f, 10.0f, 0.0f);`

左右の矢印キーで砲台の方位角の変更

- キーを押している間 true
  - bool ofGetKeyPressed(int key)
    - key: OF_KEY _LEFT (←), OF_KEY _RIGHT (→)
- 方位角（y 軸周りの回転角）を現在の向きから増減する
  - （角度が度の場合）void ofNode::panDeg(float degrees)
  - （角度がラジアンの場合）void ofNode::panRad(float radians)
    - 例）`turn_table.panDeg(1.0f);`

上下の矢印キーで砲身の方位角の変更

- キーを押している間 true
  - bool ofGetKeyPressed(int key)
    - key: OF_KEY _UP (↑), OF_KEY _DOWN (↓)
- 仰角（x 軸周りの回転角）を現在の向きから増減する
  - （角度が度の場合）void ofNode::tiltDeg(float degrees)
  - （角度がラジアンの場合）void ofNode::tiltRad(float radians)
    - 例）`barrel.tiltDeg(-1.0f);`

砲弾の位置と速度の更新

- 経過時間の取得
  - double ofGetLastFrameTime()
    - 例）`const float dt = ofGetLastFrameTime();`
- 位置の更新
  - void ofNode::move(const glm::vec3 &offset)
    - 例）`bullet.move(velocity * dt);`
- 速度の更新
  - $\mathbf{v}' = \mathbf{v} + \mathbf{a}t$
    - 例）`velocity += gravity * dt;`

砲弾の着弾判定

- 位置の取得
  - glm::vec3 ofNode::getPosition()
  - float ofNode::getX(), float ofNode::getY(), float ofNode::getZ()
- 球の半径の取得
  - float ofSpherePrimitive::getRadius()
- 地面への落下の判定
  - 砲弾の高さ y が砲弾の半径以下のときに { … } 内の処理を行う
    - 例）`if (bullet.getY() <= bullet.getRadius()) { … }`
  - { … } 内の処理で何をするかは自分で考えてください

目標物との衝突の判定

- ２点間の距離を求める
  - T glm::distance(const vecType &p0=P, const vecType &p1=P)
    - 例）`float d = distance(bullet.getPosition(), target.getPosition());`
  - 砲弾と目標物との距離が一定以下の時に { … } の処理を行う
    - 例）`if (distance(bullet.getPosition(), target.getPosition()) <= bullet.getRadius() + target の大きさ) { … }`
  - target（目標物）は自分でプログラムに追加してください
  - { … } 内の処理で何をするかは自分で考えてください

シーンの描画処理

- カメラの使用開始
  - void ofCamera::begin()
    - 例）`camera.begin()`
- カメラの使用終了
  - void ofCamera::end()
    - 例）`camera.end()`
- 図形（of3dPrimitive クラス）の描画
  - void of3dPrimitive::draw()
    - 例）`bullet.draw();`

角度の取得

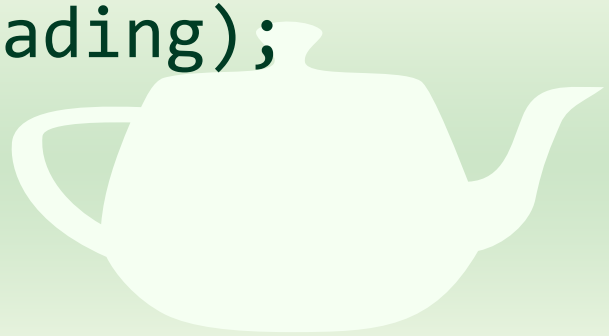
- 方位角の取得
  - （度で取得）float ofNode::getHeadingDeg()
  - （ラジアンで取得）float ofNode::getHeadingRad()
    - 例）`const float heading = turn_table.getHeadingRad();`
- 仰角の取得
  - （度で取得）float ofNode::getPitchDeg()
  - （ラジアンで取得）float ofNode::getPitchRad()
    - 例）`const float pitch = barrel.getPitchRad();`

方位角と仰角から方向ベクトルを算出

- θ: 方位角、ϕ: 仰角、$\mathbf{d} = (x \quad y \quad z)^{\mathsf{T}}$: 方向ベクトル

$$x = \sin\phi \sin\theta$$
$$y = \cos\phi$$
$$z = \sin\phi \cos\theta$$

  - 例）
```
const float x = sin(pitch) * sin(heading);
const float y = cos(pitch);
const float z = sin(pitch) * cos(heading);
```



速度と加速度（重力加速度）の設定

- $\mathbf{v}$: 速度ベクトル、 v: 速度の絶対値、$\mathbf{d} = (x \quad y \quad z)^\intercal$: 方向ベクトル

$$\mathbf{v} = v\mathbf{d}$$

  - 例）`velocity = 50.0f * vec3{ x, y, z };`
- $\mathbf{a} = (a_x \quad a_y \quad a_z)^\intercal$: 加速度ベクトル、$g = -9.8$: 重力加速度

$$\mathbf{a} = (0 \quad g \quad 0)^\intercal$$

  - 例）`gravity = vec3{ 0.0f, -9.8f, 0.0f };`

目標物の設定

ランダムな位置に目標物を配置する

目標物の作成

- 適当な図形のクラスのオブジェクトを生成する
  - ofBoxPrimitive クラス等
    - 例）`ofBoxPrimitive target;`

目標物の配置

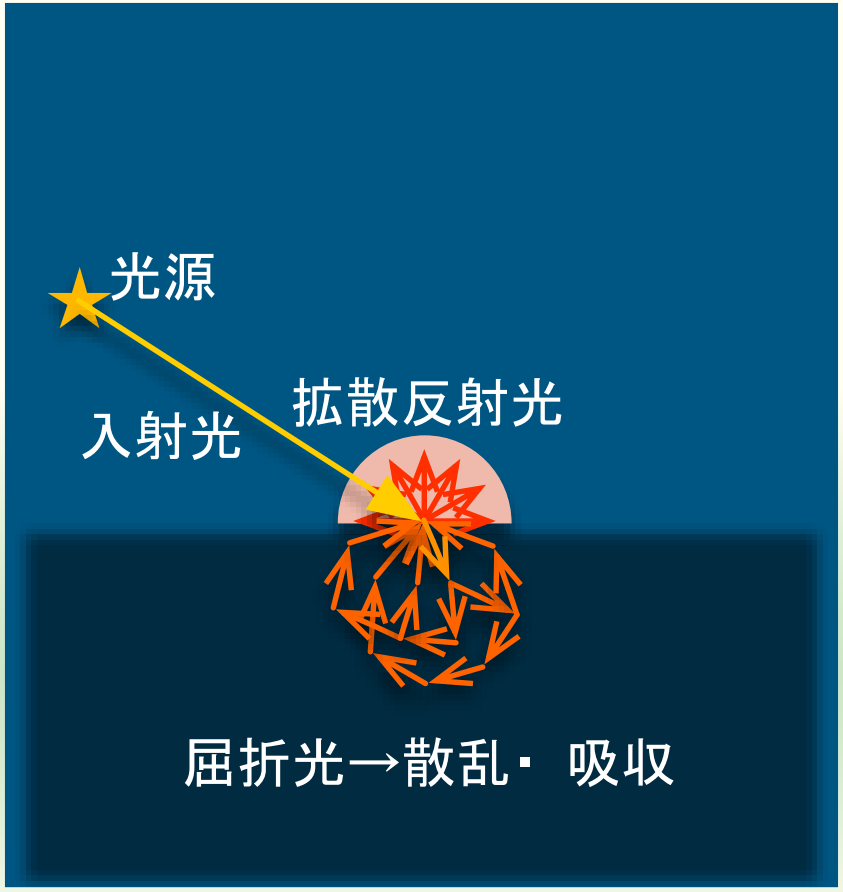
- 地面の領域内の任意の位置を乱数で決める
  - 地面の幅 w、奥行 d、中心位置 p
    - `float w = grand.getWidth(), d = grand.getHeight();`
    - `vec3 p = grand.getPosition();`
  - 目標物の位置の座標値 (x, y = 0, z)
    - `float x = ofRandom(-0.5f * w + p.x, 0.5f * w + p.x);`
    - `float z = ofRandom(-0.5f * d + p.z, 0.5f * d + p.z);`
- 何かのタイミングで目標物に位置を設定する
  - 起動時、着弾時、何かのキーをタイプしたときなど
    - 例）`target.setPosition(x, 0.0f, z);`

材質の設定

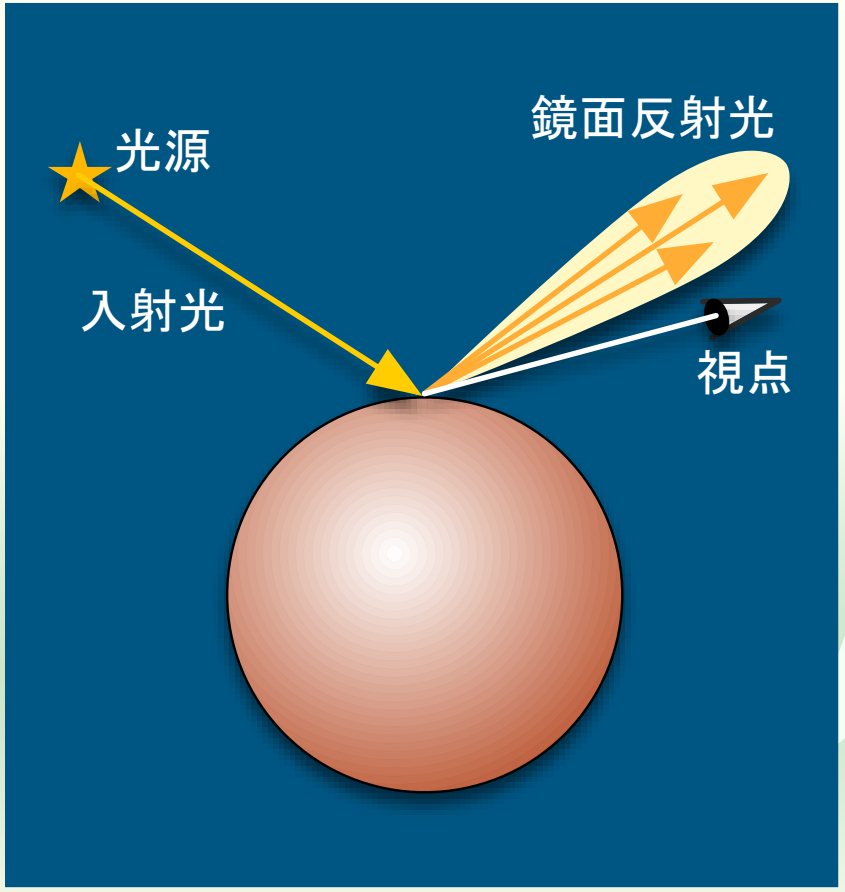
ofMaterial クラス

二色性陰影付けモデル

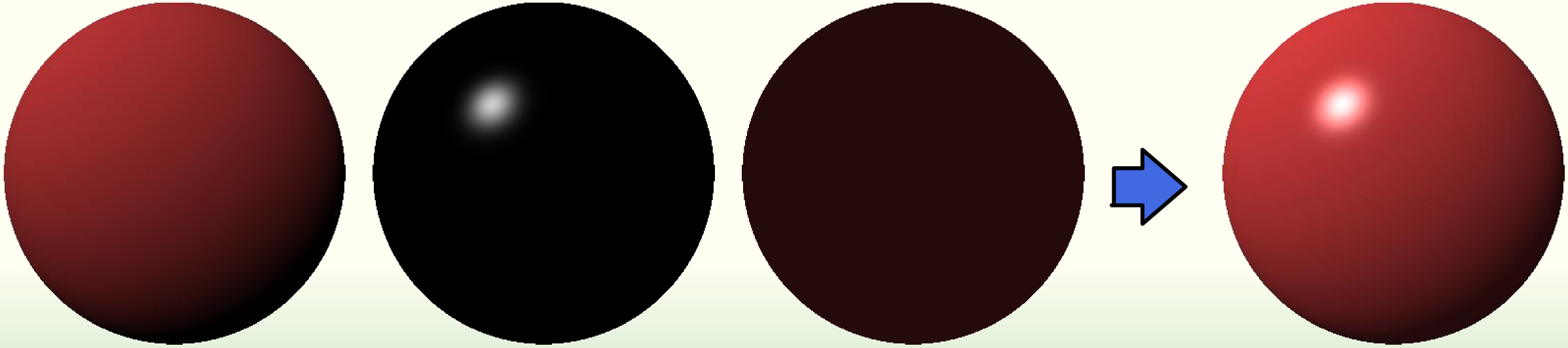
拡散反射光 (diffuse)



鏡面反射光 (specular)



陰影付け方程式



拡散反射光

I_{diff}

鏡面反射光

I_{spec}

環境光の反射光

I_{amb}

反射光強度

$$I_{tot} = I_{diff} + I_{spec} + I_{amb}$$

材質の設定と描画

ofApp.h 等

```
// 材質
ofMaterial bullet_material;
```

setup() 等

```
// 拡散反射係数と鏡面反射係数
const ofFloatColor diffuse{ 0.8f, 0.2f, 0.4f };
const ofFloatColor specular{ 0.3f, 0.3f, 0.3f };

// 輝き係数
const float shininess = 30.0f;

// 材質の設定
bullet_material.setAmbientColor(diffuse);
bullet_material.setDiffuseColor(diffuse);
bullet_material.setSpecularColor(specular);
bullet_material.setShininess(shininess);
```

エネルギー保存の法則に従うなら上下を足して 1 を超えたらまずい

diffuse と ambient の色は同じにするのが基本

draw() 等

```
// 描画
bullet_material.begin();
bullet.draw();
bullet_material.end();
```

- ofMaterial クラスのオブジェクトを生成する
- 拡散反射係数、鏡面反射係数、および輝き係数を決定する
  - 輝き係数が大きいほどハイライトは小さくなる
  - 非金属の材質では鏡面反射係数はグレーにしておくのが基本
  - 金属・半導体では鏡面反射光にも色が付く場合がある
- begin()～end() の間で描画する