

ゲームグラフィックス特論

第13回 遅延レンダリング

Render To Texture

テクスチャにレンダリング

レンダリング結果を素材として利用する

- 映り込みや屈折などの光学的効果
 1. 視点を変更してレンダリングする
 2. レンダリング結果をテクスチャとしてマッピングする
 - このレンダリング結果は直接には画面に表示されない
- 素材を作成するために画面表示を行わずにレンダリングする
 - オフスクリーンレンダリング
 - かつては
 - フレームバッファの表示領域外
 - ダブルバッファリング時のバックバッファ, など
 - レンダリング結果をテクスチャメモリに転送する必要がある

フレームバッファオブジェクト

- 標準のフレームバッファ
 - あらかじめ用意されている
 - 描き込んだ図形はディスプレイに表示される
- フレームバッファオブジェクト
 - Frame Buffer Object, FBO
 - ユーザが自分で用意する
 - 図形を描き込んでもディスプレイには表示されない
 - テクスチャとして直接参照できる
- FBO を使えばテクスチャメモリに直接レンダリングできる



Render To Texture

フレームバッファオブジェクトの作成

- フレームバッファは複数のバッファの集合体
 - カラーバッファ（色）
 - デプスバッファ（深度）
 - ステンシルバッファ（型抜き）
- それぞれのバッファとして使うメモリを確保する
 - テクスチャにより確保する
 - レンダリング結果をテクスチャとして利用する場合
 - レンダーバッファにより確保する
 - レンダリング結果をテクスチャとして利用しない場合
- 確保したメモリを組み合わせるフレームバッファを構成する
 - 使わないバッファのメモリは準備する必要は無い
 - 準備しないバッファへの読み書きは禁止しておく

カラーバッファ用テクスチャの確保

```
#define FBOWIDTH  512          // フレームバッファオブジェクトの幅
#define FBOHEIGHT 512          // フレームバッファオブジェクトの高さ

...

GLuint cb;                     // カラーバッファ用のテクスチャ名

...

glGenTextures(1, &cb);
glBindTexture(GL_TEXTURE_2D, cb);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA,
             FBOWIDTH, FBOHEIGHT, 0,
             GL_RGBA, GL_UNSIGNED_BYTE, nullptr);
             これは多分どうでもいい

... (その他のテクスチャ設定)
glBindTexture(GL_TEXTURE_2D, 0);
```

GL_RGBAだと $[0,1]$ にクランプされる
GL_RGBA32F なら実数（単精度）が使える（浮動小数点テクスチャ）

レンダリングによって書き込むので
ここで画像を読み込む必要は無い

デプスバッファ用テクスチャの確保

```
Gluint db; // デプスバッファ用のテクスチャ名
```

```
...
```

```
// デプスバッファ用のテクスチャを用意する
```

```
glGenTextures(1, &db);  
glBindTexture(GL_TEXTURE_2D, db);  
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT,  
             FBOWIDTH, FBOHEIGHT, 0,  
             GL_DEPTH_COMPONENT, GL_UNSIGNED_BYTE, nullptr);
```

デプスバッファなので

これも多分どうでもいい

```
... (その他のテクスチャ設定)
```

```
glBindTexture(GL_TEXTURE_2D, 0);
```

フレームバッファオブジェクトの作成

```
GLuint fb;                                // フレームバッファオブジェクト名
...

// フレームバッファオブジェクトを作成する
glGenFramebuffers(1, &fb);
glBindFramebuffer(GL_FRAMEBUFFER, fb);

// カラーバッファとしてテクスチャを結合する
glFramebufferTexture2D(GL_FRAMEBUFFER,
    GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, cb, 0);

// デプスバッファとしてテクスチャを結合する
glFramebufferTexture2D(GL_FRAMEBUFFER,
    GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, db, 0);

// フレームバッファオブジェクトの結合を解除する
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```


レンダーバッファをデプスバッファに使う

```
GluInt rb;                                // デプスバッファ用のレンダーバッファ名
...

// デプスバッファ用のレンダーバッファを用意する
glGenRenderbuffers(1, &rb);
glBindRenderbuffer(GL_RENDERBUFFER, rb);
glRenderbufferStorage(GL_RENDERBUFFER,
    GL_DEPTH_COMPONENT, FBOWIDTH, FBOHEIGHT);

// レンダーバッファの結合を解除する
glBindRenderbuffer(GL_RENDERBUFFER, 0);
```

フレームバッファオブジェクトの作成

```
// フレームバッファオブジェクトを作成する
glGenFramebuffers(1, &fb);
glBindFramebuffer(GL_FRAMEBUFFER, fb);

// カラーバッファとしてテクスチャを結合する
glFramebufferTexture2D(GL_FRAMEBUFFER,
    GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, cb, 0);

// デプスバッファとしてレンダーバッファを結合する
glFramebufferRenderbuffer(GL_FRAMEBUFFER,
    GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER, rb);

// フレームバッファオブジェクトの結合を解除する
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

カラーバッファを使わない場合

```
// フレームバッファオブジェクトを作成する
glGenFramebuffers(1, &fb);
glBindFramebuffer(GL_FRAMEBUFFER, fb);

// フレームバッファオブジェクトにデプスバッファのテクスチャを結合する
glFramebufferTexture2D(GL_FRAMEBUFFER,
    GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, db, 0);

// カラーバッファは無いので読み書きしない
glDrawBuffer(GL_NONE);
glReadBuffer(GL_NONE);

// フレームバッファオブジェクトの結合を解除する
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

シャドウマッピングに使う
デプステクスチャ作成など

フレームバッファオブジェクトへの描画

```
// フレームバッファオブジェクトを結合する
glBindFramebuffer(GL_FRAMEBUFFER, fb);

// ビューポートはフレームバッファオブジェクトのサイズ以下にする
glViewport(0, 0, FBOWIDTH, FBOHEIGHT);

// シーンの描画
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

...

glFlush(); // FBO への描き込み完了を待つなら glFinish();

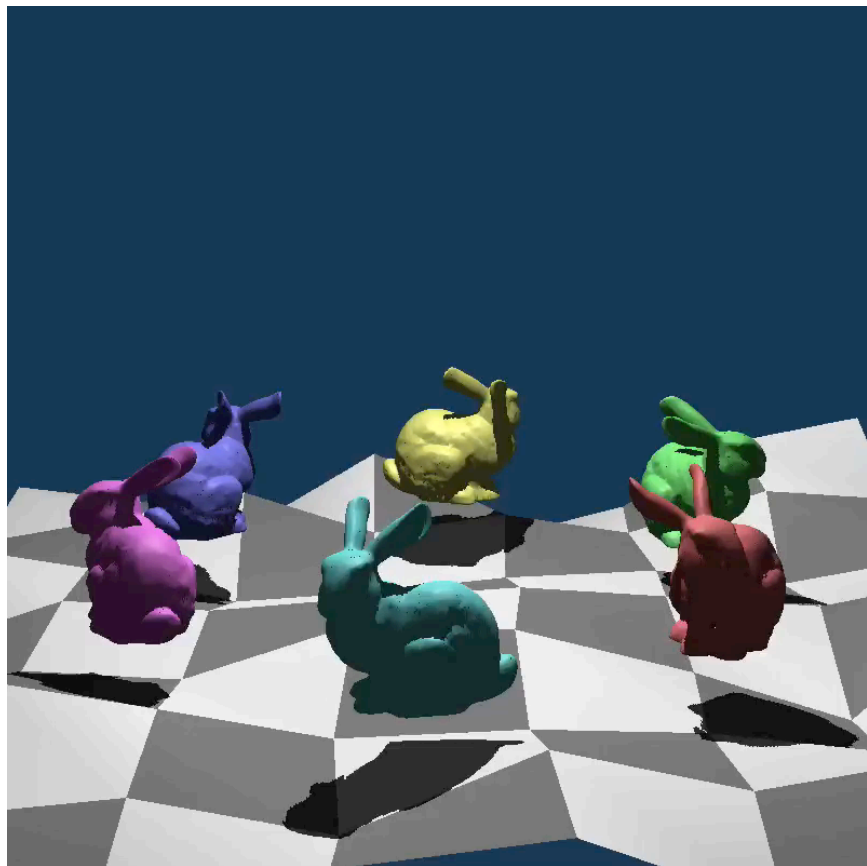
// フレームバッファオブジェクトの結合を解除する
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

応用例

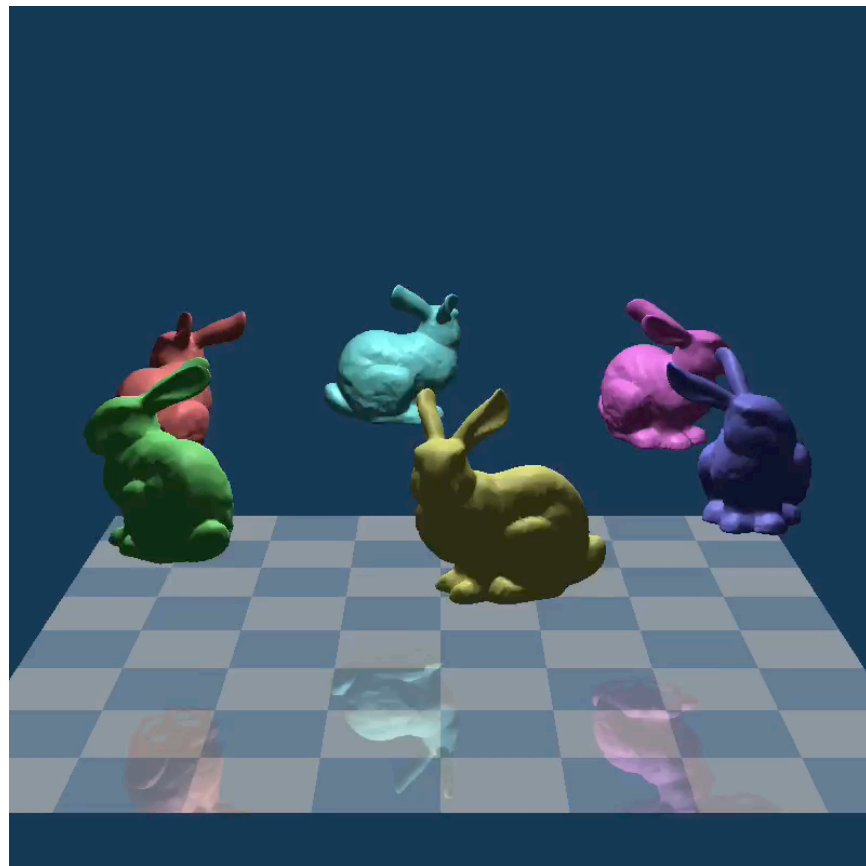
FBO の使い方の実例

応用例

シャドウマッピング



映り込み処理



シャドウマッピング

1. フレームバッファオブジェクトを作成する
 - カラーバッファは用いない／読み書きしない
 - デプスバッファはテクスチャ（デプスマップ）
2. 描画先をフレームバッファオブジェクトに切り替える
3. 光源側から見たシーンを描く
 - 使用した投影変換行列とモデルビュー変換行列を保存しておく
4. 描画先を通常のフレームバッファに戻す
5. 視点側から見たシーンを描く
 - テクスチャをデプスマップとしてマッピングする

映り込み処理 (宿題のヒントに実装例)


1. フレームバッファオブジェクトを作成する
 - カラーバッファはテクスチャ
 - デプスバッファはレンダーバッファ
2. 描画先をフレームバッファオブジェクトに切り替える
3. 図形の鏡像をフレームバッファオブジェクトに描く
4. 描画先を通常のフレームバッファに戻す
5. 本来の図形を描く
6. 床を描く
 - フレームバッファオブジェクトのカラーバッファに使ったテクスチャをマッピングする

Multiple Render Target

一度に複数の画像を生成する

マルチプルレンダーターターゲット

- フレームバッファオブジェクト
 - カラーバッファを複数持たせることができる
- マルチプルレンダーターターゲット
 - フラグメントシェーダが同時に複数のカラーバッファに描き込む機能
- 最終的なレンダリング結果は複数の画像を合成したもの
 - 環境光の反射光, 拡散反射光, 鏡面反射光
 - 映り込み, 透過光
 - 物体表面の座標値, 物体表面の法線ベクトル
 - 物体の識別子, 材質の識別子
 - ...
- マルチパスレンダリング
 - かつてはレンダリング結果をフレームバッファ上で合成していた



色情報ではないもの

浮動小数点テクスチャ

複数のカラーバッファの準備

```
GLint cb, sb, ab;

...
// 拡散反射光を格納するテクスチャを用意する
glGenTextures(1, &cb);
glBindTexture(GL_TEXTURE_2D, cb);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, (中略), nullptr);

...
// 鏡面反射光を格納するテクスチャを用意する
glGenTextures(1, &sb);
glBindTexture(GL_TEXTURE_2D, sb);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, (中略), nullptr);

...
// 環境光の反射光を格納するテクスチャを用意する
glGenTextures(1, &ab);
glBindTexture(GL_TEXTURE_2D, ab);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, (中略), nullptr);

...

glBindTexture(GL_TEXTURE_2D, 0);
```

フレームバッファオブジェクトの作成

```
// フレームバッファオブジェクトを作成
glGenFramebuffers(1, &fb);
glBindFramebuffer(GL_FRAMEBUFFER, fb);

// フレームバッファオブジェクトに拡散反射光の格納先のテクスチャを結合
glFramebufferTexture2D(GL_FRAMEBUFFER,
    GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, cb, 0);

// フレームバッファオブジェクトに鏡面反射光の格納先のテクスチャを結合
glFramebufferTexture2D(GL_FRAMEBUFFER,
    GL_COLOR_ATTACHMENT1, GL_TEXTURE_2D, sb, 0);

// フレームバッファオブジェクトに環境光の反射光の格納先のテクスチャを結合
glFramebufferTexture2D(GL_FRAMEBUFFER,
    GL_COLOR_ATTACHMENT2, GL_TEXTURE_2D, ab, 0);

...

// フレームバッファオブジェクトの結合を解除
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

レンダーターゲット

```
/*
** レンダーターゲットのリスト
*/
const GLenum bufs[] =
{
    GL_COLOR_ATTACHMENT0,           // カラーバッファ（拡散反射光）
    GL_COLOR_ATTACHMENT1,           // 鏡面反射光
    GL_COLOR_ATTACHMENT2,           // 環境光の反射光
};
```

- デフォルトのフレームバッファ（GL_FRONT_LEFT, GL_FRONT_RIGHT, GL_BACK_LEFT, GL_BACK_RIGHT）と GL_COLOR_ATTACHMENT_n とを混在して指定することはできない
- 複数のバッファを表す GL_FRONT, GL_BACK, GL_LEFT, GL_RIGHT, GL_FRONT_AND_BACK を指定することはできない

レンダーターゲットの指定

```
// フレームバッファオブジェクトを結合する
glBindFramebuffer(GL_FRAMEBUFFER, fb);

// レンダーターゲットを指定する
glDrawBuffers(sizeof bufs / sizeof bufs[0], bufs);
               レンダーターゲットの数 (bufs の要素数)

// シーンの描画
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
...
glFlush();

// レンダーターゲットを元に戻す
glDrawBuffer(GL_BACK);           // ダブルバッファリングのとき

// フレームバッファオブジェクトの結合を解除する
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

フラグメントシェーダの例

```
#version 150 core
#extension GL_ARB_explicit_attrib_location : enable
...

in vec4 iamb;
in vec4 idiff;
in vec4 ispec;

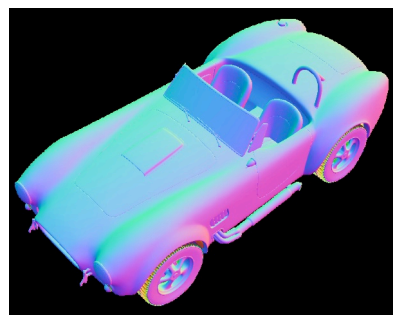
out (layout = 0) vec4 diffuse;
out (layout = 1) vec4 specular;
out (layout = 2) vec4 ambient;

void main()
{
    diffuse = idiff;
    specular = ispec;
    ambient = iamb;
}
```

GLSL 3.3
(#version 330)
以降では不要

// 環境光の反射光 A
// 拡散反射光 D
// 鏡面反射光 S

// bufs[0] に指定したバッファ
// bufs[1] に指定したバッファ
// bufs[2] に指定したバッファ



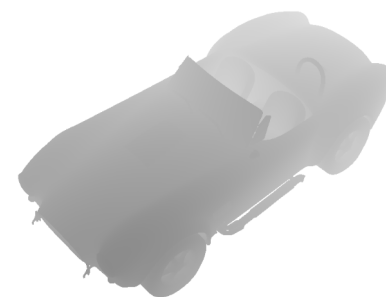
法線ベクトル



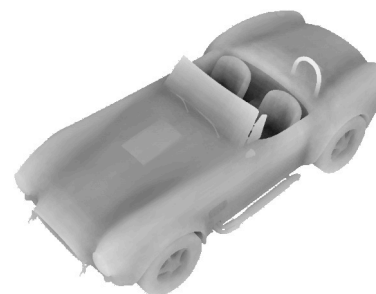
放射照度マップ



放射照度



深度



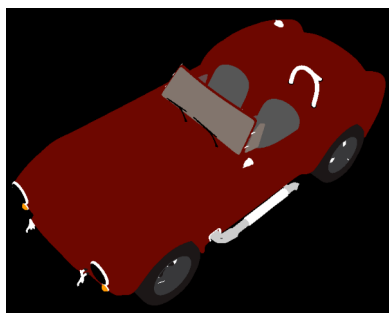
環境遮蔽

環境遮蔽を考慮した
放射照度

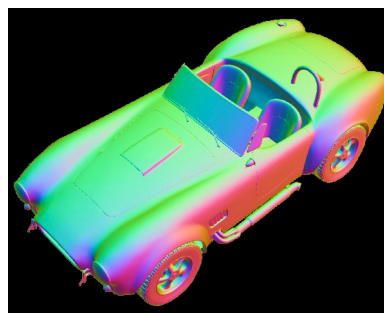
MRT を使った 遅延レンダリング



環境遮蔽を考慮した
放射照度



アルベド



視線の反射ベクトル



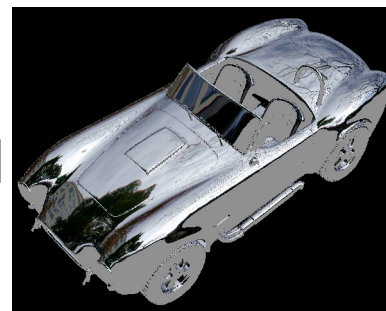
環境マップ



拡散反射光強度



フレネル係数



映り込み



合成結果

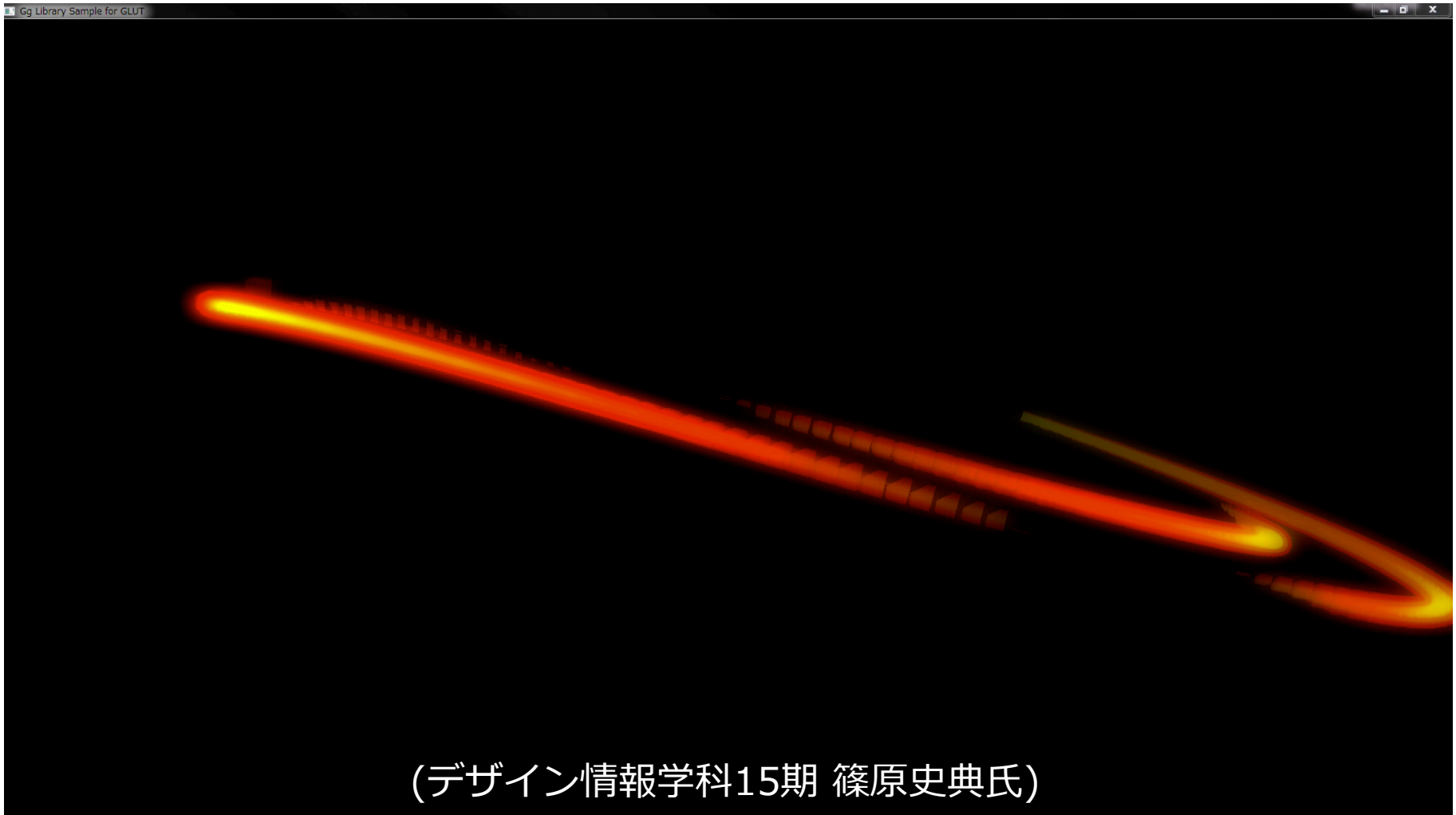
Post Processing Effect

画像処理による映像効果

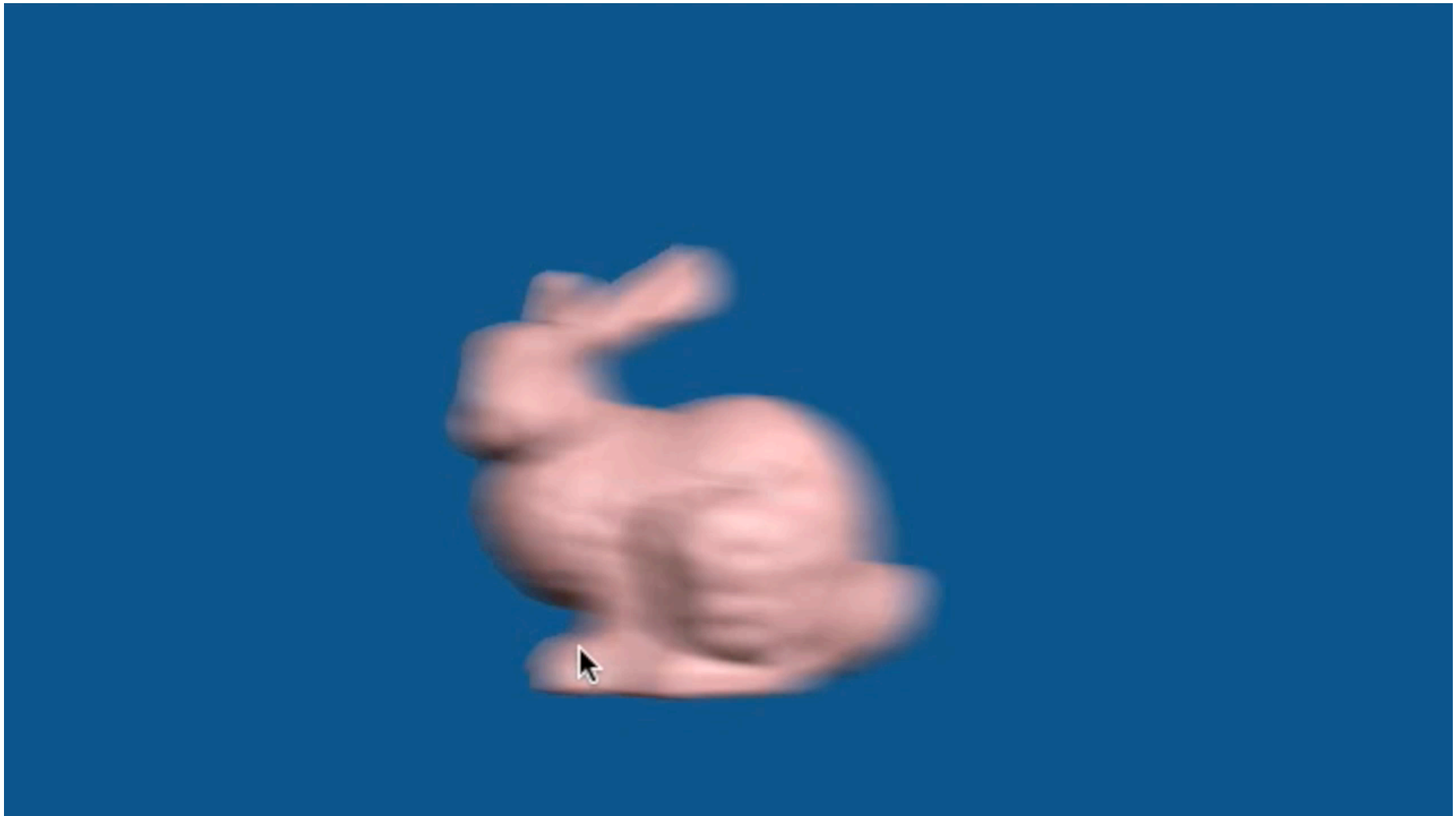
遅延レンダリング

- ある種の映像効果はレンダリング後に適用する
 - Screen Space Ambient Occlusion (SSAO)
 - グロー効果・ブルーム効果
 - モーションブラー
 - Motion Blur as a Post-Processing Effect (GPU Gems 3, Chapter 27)
 - ある種のアンチエイリアシング手法
 - Morphological Antialiasing (MLAA)
 - Reshetov, Alexander. "Morphological antialiasing." *Proceedings of the Conference on High Performance Graphics 2009*. ACM, 2009.
 - Fast Approximate Anti Aliasing (FXAA)
 - Lottes, T. "Fast approximate anti-aliasing." (2009).
 - Subpixel Morphological Antialiasing (SMAA)
 - Jimenez, Jorge, et al. "SMAA: Enhanced subpixel morphological antialiasing." *Computer Graphics Forum*. Vol. 31. No. 2pt1. Blackwell Publishing Ltd, 2012.
- 複雑な光学効果による陰影を可視面に対してのみ行う
 - デプスバッファ法による隠面消去処理で捨てられてしまう無駄を避ける

残光エフェクト



Screen Space Motion Blur



遅延レンダリングの手順

- テクスチャメモリに画像を用意する
 - 画像の読み込み
 - Render To Texture
 - Multiple Render Target
- クリッピング空間を覆う 1 枚のポリゴンを描く
 - バーテックスシェーダ
 - ほとんど何もしない
 - クリッピング座標 $(-1, -1)-(1, 1)$ からテクスチャ座標 $(0, 0)-(1, 1)$ を生成する
 - フラグメントシェーダ
 - バーテックスシェーダから渡されたテクスチャ座標でテクスチャをサンプリングして画像処理を行う

通常のフレームバッファへに切り替える

...

```
// フレームバッファオブジェクトの結合を解除する  
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

```
// レンダーターゲットを元に戻す  
glDrawBuffer(GL_BACK);    // シングルバッファリングなら GL_FRONT
```

```
// ビューポートを表示領域のサイズに合わせる  
glViewport(0, 0, width, height);
```

```
// 隠面消去処理は行わない  
glDisable(GL_DEPTH_TEST);
```

先に保存しておくか OS あるいは
ウィンドウシステムから取得する

1枚のポリゴンを描く

```
// クリッピング空間いっぱいの矩形ポリゴン
```

```
static const GLfloat pv[][2] = {  
    { -1.0f, -1.0f },  
    {  1.0f, -1.0f },  
    {  1.0f,  1.0f },  
    { -1.0f,  1.0f },  
};
```

```
// 表示領域を覆うポリゴンを描く
```

```
GLuint vao;  
glGenVertexArrays(1, &vao);  
glBindVertexArray(vao);
```

```
GLuint vbo;  
glGenBuffers(1, &vbo);  
glBindBuffer(vbo);  
glBufferData(GL_ARRAY_BUFFER, sizeof pv, pv, GL_STATIC_DRAW);  
glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 0, 0);  
glEnableVertexAttribArray(0);  
glDrawArrays(GL_TRIANGLE_FAN, 0, sizeof pv / sizeof pv[0]);
```


バーテックスシェーダ

```
#version 150 core

in vec2 pv;           // 矩形の頂点位置
out vec2 tc;          // フラグメントシェーダに送るテクスチャ座標

void main()
{
    tc = pv * 0.5 + 0.5; // (-1,-1)-(1,1)→(0,0)-(1,1)の変換
    gl_Position = vec4(pv, 0.0, 1.0);
}
```

フラグメントシェーダーの例

```
#version 150 core

uniform sampler2D diffuse; // 拡散反射光のテクスチャユニット
uniform sampler2D specular; // 鏡面反射光のテクスチャユニット
uniform sampler2D ambient; // 環境光のテクスチャユニット

in vec2 tc; // 補間されたテクスチャ座標
out vec4 fc; // フラグメントの色

void main()
{
    vec4 d = texture(diffuse, tc);
    vec4 s = texture(specular, tc);
    vec4 a = texture(ambient, tc);

    fc = d + s + a; // 足してるだけ
}
```

輪郭強調フィルタ（4近傍）

```
#version 150 core
```

```
uniform sampler2D diffuse; // 拡散反射光のテクスチャユニット  
uniform sampler2D specular; // 鏡面反射光のテクスチャユニット  
uniform sampler2D ambient; // 環境光のテクスチャユニット
```

```
in vec2 tc; // 補間されたテクスチャ座標  
out vec4 fc; // フラグメントの色
```

```
void main()  
{
```

```
    vec4 d = (texture(diffuse, tc) * 5.0  
              - texture0ffset(diffuse, tc, ivec2( 0, -1))  
              - texture0ffset(diffuse, tc, ivec2(-1,  0))  
              - texture0ffset(diffuse, tc, ivec2( 1,  0))  
              - texture0ffset(diffuse, tc, ivec2( 0,  1))) * 0.25;  
    fc = d + texture(specular, tc) + texture2d(ambient, tc);
```

```
}
```

0	-1	0
-1	5	-1
0	-1	0

宿題

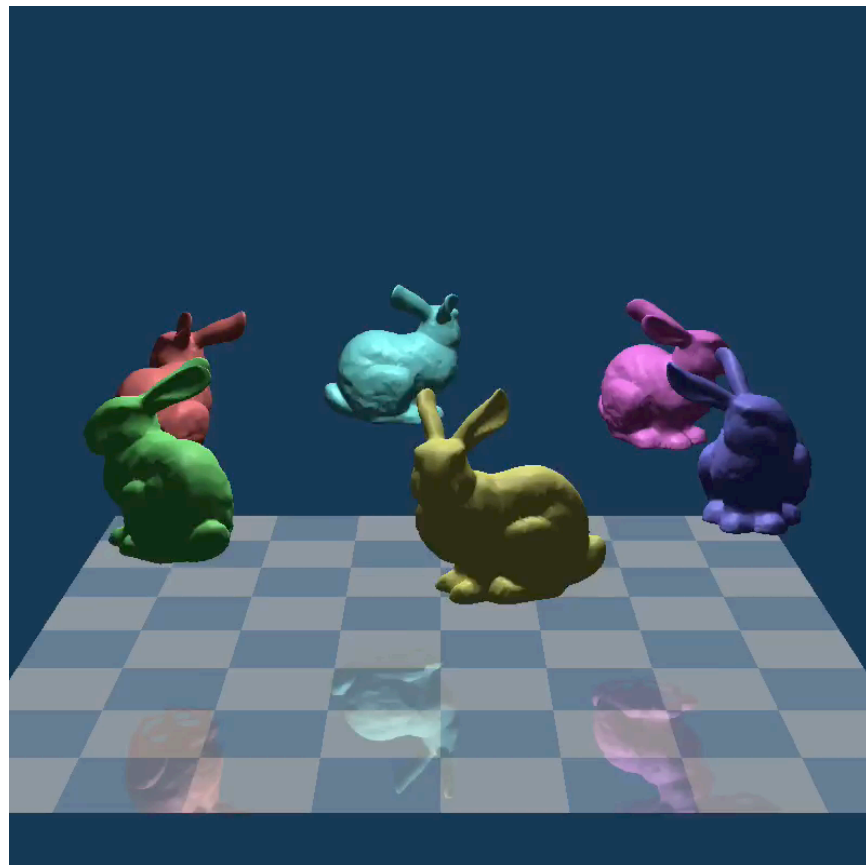
- FBO を使って平面への映り込み処理を実装してください。
 - 次のプログラムは市松模様のついた平面の上を物体が移動します。
 - <https://github.com/tokoik/ggsample13>
 - この市松模様のついた平面に物体が映り込むようにしてください。
 - プログラムの各所を調べればヒントがあるかもしれません。
 - 変更するのは `ggsample13.cpp` と `ggsample13tile.vert` / `ggsample13tile.frag` だけです (多分) .
- `ggsample13.cpp`, `ggsample13tile.vert` , `ggsample13tile.frag` を**アップロード**してください
 - アップロード先
 - <https://www.wakayama-u.ac.jp/~tokoi/lecture/gg/upload/>
 - 根性があったら映り込みをぼかしてください。
 - さらにシャドウマップ法を使って影を付けてください。
 - シャドウマップ法にも FBO を使ってください。

宿題プログラムの生成画像

元のプログラム



期待する結果



映り込み処理の実装のヒント

- 鏡像は元のオブジェクトの上下 (y 軸) を反転します
 - これによりポリゴンの表裏が反転しますので, 正しく陰影付けするには法線ベクトルを反転する必要があります
 - ggsample13.vert をコピーして鏡像用のバーテックスシェーダ ggsample13mirror.vert を作成し法線ベクトル n を反転 (負号をつける)

```
// 正像用のプログラムオブジェクト
```

```
GgSimpleShader simple("ggsample13.vert", "ggsample13.frag");
```

```
// 鏡像用のプログラムオブジェクト
```

```
GgSimpleShader mirror("ggsample13mirror.vert", "ggsample13.frag");
```

$\max(\text{dot}(n, l), 0.0)$ を $\text{abs}(\text{dot}(n, l))$ にすれば simple.vert と共用にできるけど...

- 背面カリングも反転 (表面をカリング) します

```
glCullFace(GL_FRONT);
```

カラーバッファのテクスチャの作成

```
// フレームバッファオブジェクトの解像度
const GLsizei fboWidth(1024), fboHeight(1024);
```

.....
この場合はウィンドウのサイズを気にしなくてよい

```
// カラーバッファ用のテクスチャを用意する
```

```
GLuint cb;
glGenTextures(1, &cb);
glBindTexture(GL_TEXTURE_2D, cb);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, fboWidth, fboHeight, 0,
             GL_RGBA, GL_UNSIGNED_BYTE, nullptr);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
                GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
                GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

レンダーバッファの作成

```
// デプスバッファ用のレンダーバッファを用意する
GLuint rb;
glGenRenderbuffers(1, &rb);
glBindRenderbuffer(GL_RENDERBUFFER, rb);
glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT,
    fboWidth, fboHeight);
```


フレームバッファオブジェクトの作成

```
// 映り込み用のフレームバッファオブジェクトを作成する
GLuint fb;
glGenFramebuffers(1, &fb);
glBindFramebuffer(GL_FRAMEBUFFER, fb);

// FBO のカラーバッファにテクスチャを取り付ける
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,
    GL_TEXTURE_2D, cb, 0);

// FBO のデプスバッファにレンダーバッファを取り付ける
glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT,
    GL_RENDERBUFFER, rb);
```

鏡像の変換行列とその光源位置を求める

```
// 正像のビュー変換行列を mv に求める
const GgMatrix mv(ggLookat(0.0f, 3.0f, 8.0f, 0.0f, 1.0f, 0.0f,
    0.0f, 1.0f, 0.0f));

// 視点座標系の光源位置を求める
mv.projection(lightProperty.position, normal);

// 鏡像のビュー変換行列を mr に求める
const GgMatrix mr(mv * ggScale(1.0f, -1.0f, 1.0f));

// 鏡像の視点座標系における光源位置
GLfloat reflect[4];
mr.projection(reflect, normal);
```

FBO にレンダリング開始

```
// ビューポートをフレームバッファオブジェクトのサイズに合わせる  
glViewport(0, 0, fboWidth, fboHeight);  
  
// フレームバッファオブジェクトを結合する  
glBindFramebuffer(GL_FRAMEBUFFER, fb);  
  
// フレームバッファオブジェクトの画面消去  
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  
  
// 鏡像用のシェーダの選択  
mirror.use();  
mirror.selectLight(light.get());  
light->loadLightPosition(reflect);
```

FBO に鏡像をレンダリング

```
// 前面をカリングする
glCullFace(GL_FRONT);

// 鏡像をフレームバッファオブジェクトに描画
drawObjects(mirror, mp, mr, object.get(),
    material.get(), objects, t);

// 背面のカリングに戻る
glCullFace(GL_BACK);

// フレームバッファオブジェクトの結合を解除する
glBindFramebuffer(GL_FRAMEBUFFER, 0);

// ビューポートをウィンドウのサイズに戻す
glViewport(0, 0, window.getWidth(), window.getHeight());
```

正像をレンダリング

```
// 画面消去
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

// 正像用のシェーダの選択
simple.use();
simple.selectLight(light.get());
light->loadLightPosition(normal);

// 正像の描画
drawObjects(simple, mp, mv, object.get(),
    material.get(), objects, t);
```

床面はカラーテクスチャを参照して描画

```
// 床面用のシェーダの選択
floor.use();
floor.selectLight(light.get());

// 床面の描画
floor.selectMaterial(tile.get());
floor.loadMatrix(mp, mv.rotateX(-1.5707963f));
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, cb);
rectangle->draw();
```

拡張課題

映り込みをぼかす



さらに影を追加する

