コンピュータグラフィックス特論 Ⅱ

第5回　影の表現（高度な描画技術）

九州工業大学　尾下 真樹

2021年度

今日の内容

- 影の描画のプログラム
- テクスチャマッピングによる影の描画
- 平面へのポリゴン投影による影の描画
- シャドウ・ヴォリューム
- シャドウ・マッピング
- 高度な影の描画技術
- 高度な描画技術
- レポート課題

影の表現

- レンダリング画像の現実感(リアリティ)を出す上で、影の描画は不可欠
  - 影の有無は、画面の自然さに大きく影響
  - 特に空中に浮いている物体を描画するようなときには、影があると、高さが把握しやすい
- 影の描画の技術
  - いくつかの方法が利用されている
  - 高度な描画技術が必要となる
    - アルファブレンディング(半透明描画)
    - ステンシルバッファ

今日の内容

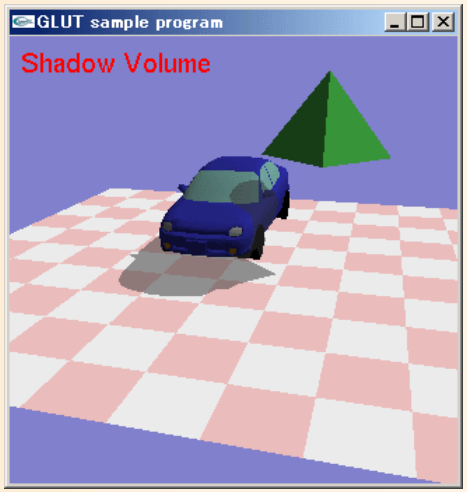
- 影の表現方法
  - テクスチャマッピング
  - 平面へのポリゴン投影
  - シャドウ・ヴォリューム
  - シャドウ・マッピング
- OpenGLの高度な描画技術
  - アルファブレンディング
  - ステンシルバッファ
- 高度な影の描画技術
  - セルフ・シャドウ、ソフト・シャドウ

今日の内容

- 影の描画のプログラム
- テクスチャマッピングによる影の描画
- 平面へのポリゴン投影による影の描画
- シャドウ・ヴォリューム
- シャドウ・マッピング
- 高度な影の描画技術
- 高度な描画技術
- レポート課題

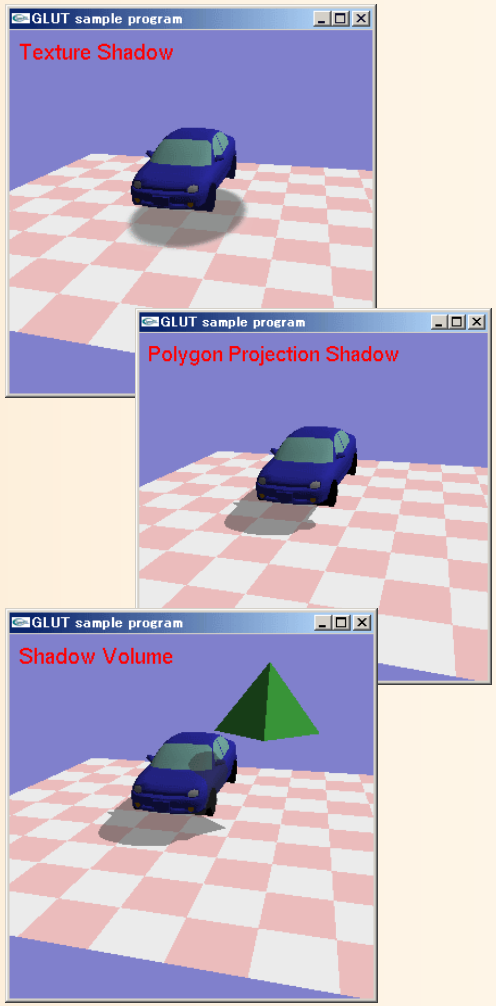
影の表現方法

- テクスチャマッピング
- 平面へのポリゴン投影
- シャドウ・ヴォリューム
- シャドウ・マッピング

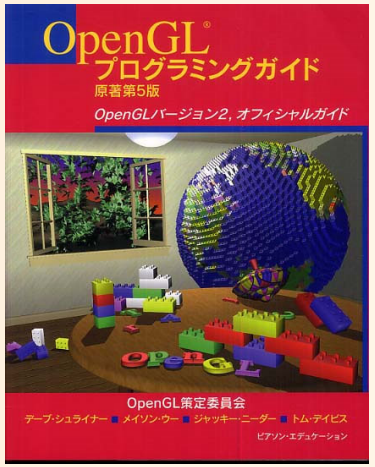


各表現方法の比較

- テクスチャマッピング
  - 高速、近似形状
  - 他の物体・自分自身への投影不可
- 平面へのポリゴン投影
  - 中速
  - 他の物体・自分自身への投影不可
- シャドウ・ヴォリューム
  - 低速
  - 他の物体への投影可、自分自身への投影可
- シャドウ・マッピング
  - 低速、ハードウェアにより高速化可能
  - 他の物体への投影可、自分自身への投影不可



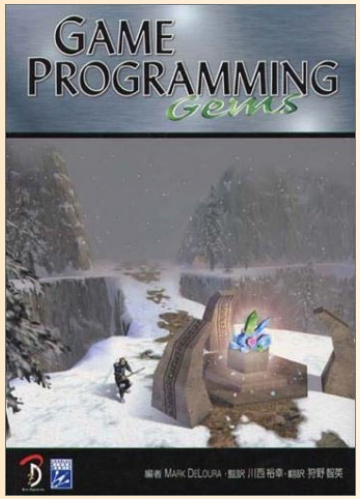
参考書

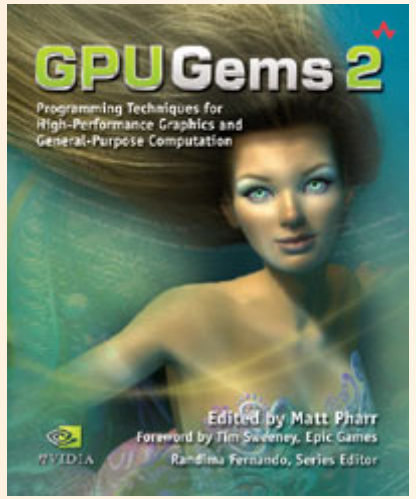


- 最低限の関数の使い方は資料を用意
- OpenGLの定番の本（高い）
  - OpenGLプログラミングガイド（赤本）, 12,000円
  - OpenGLリファレンスマニュアル（青本）, 8,300円
    - ピアソン・エデュケーション出版
- グラフィックスS（システム創成3年前期）演習資料
  - http://www.cg.ces.kyutech.ac.jp/lecture/cg/
  - OpenGLの使い方を段階的に学べるチュートリアル
  - OpenGLに不慣れな人は一通り試しておくことを推奨
- 適当な入門書
  - 他にもOpenGLの入門書は多数ある

参考資料

- テクスチャマッピング
  - 参考書(赤本)を参照
- 平面へのポリゴン投影
  - Game Programming Gems I
  - 参考書(赤本)を参照
- シャドウ・ヴォリューム
- シャドウ・マッピング
  - Game Programming Gems II
  - GPU Gems I~II
- どの方法も、ネットで検索すると参考資料が見つかる

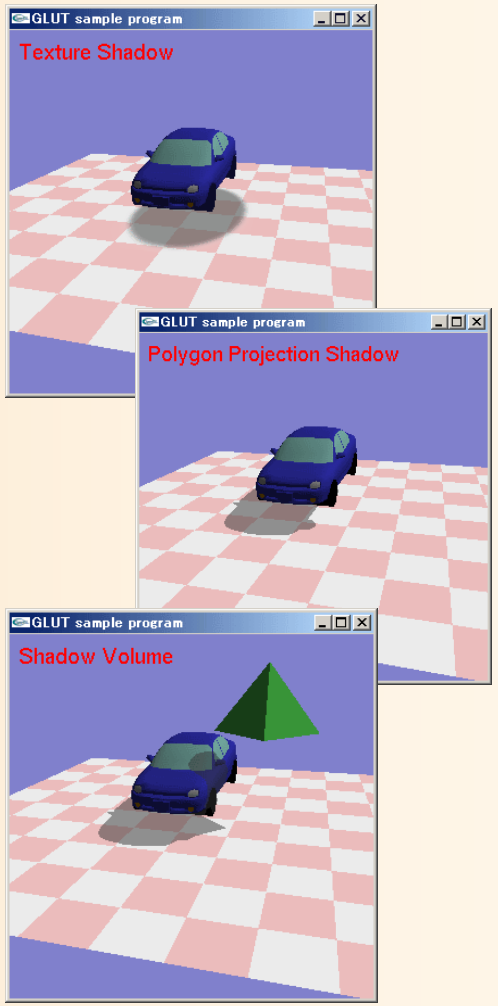




影の描画のプログラム

デモプログラム

- 影の描画
  - 3種類の方法を切り替え可能
    - テクスチャマッピング
    - 平面へのポリゴン投影
    - シャドウ・ヴォリューム
  - 物体の表示・非表示
    - 物体同士の影の確認
  - 視点操作・形状データの読み込みは、これまでの講義で扱った技術を利用




GLUT sample program
No Shadow
影の描画
Shadow

サンプルプログラムの構成

- デモプログラムの一部を実装したサンプルプログラム（shadow_sample.cpp）
  - レポート課題のもとになるプログラム
- 幾何形状の読み込み・描画（Obj.h,Obj.cpp）
  - 第4回の授業で扱った内容
- ビットマップ画像の読み込み（bitmap.h, bitmap.cpp）
  - テクスチャ画像の読み込みに使用
  - BMP（24ビット、非圧縮）形式の読み込み関数

幾何形状の定義

```
//幾何形状モデル(Obj形式)
struct  Obj
{
        int         num_vertices;   // 頂点数
        Vector *    vertices;       // 頂点座標配列

        int         num_normals;    // 法線ベクトル数
        Vector *    normals;        // 法線ベクトル配列

        int         num_textures;   // テクスチャ座標数
        Vector *    textures;       // テクスチャ座標配列

        int         num_triangles;  // 三角面数
        int *       tri_v_no;       // 三角面の頂点座標番号の配列
        int *       tri_vn_no;      // 三角面の法線ベクトル番号の配列
        int *       tri_vt_no;      // 三角面のテクスチャ座標番号の配列
        Mtl *       tri_material;   // 三角面の素材の配列
};
```

幾何形状の読み込み・描画関数

```
// Objファイルの読み込み
Obj *  LoadObj( const char * filename );

// Mtlファイルの読み込み
void  LoadMtl( const char * filename, Obj * obj );

//幾何形状モデルのスケーリング（スケーリング後のサイズを返す）
void  ScaleObj( Obj * obj, float max_size,
                float * size_x, float * size_y, float * size_z );

// Obj形状データの描画
void  RenderObj( Obj * obj );
```

サンプルプログラム(1)

- 影の描画方法や光源位置を表す変数

```
// 影の描画方法
enum  ShadowModeEnum
{
    SHADOW_NONE,
    SHADOW_TEXTURE,
    SHADOW_PROJECTION,
    SHADOW_VOLUME,
    NUM_SHADOW_MODE
};

// 現在の影の描画方法
ShadowModeEnum    shadow_mode = SHADOW_PROJECTION;

// 点光源の位置(影の投影方向)
Vector    light_pos;
```

サンプルプログラム(2)

- 幾何形状オブジェクトの情報

```
// 幾何形状オブジェクトの数
#define  NUM_OBJECTS  2

// 幾何形状オブジェクト
Obj *     object[ NUM_OBJECTS ];

// 位置
Vector    object_pos[ NUM_OBJECTS];

// 水平向き
float     object_ori[ NUM_OBJECTS ];

// 大きさ(テクスチャマッピングによる影の描画用)
Vector    object_size[ NUM_OBJECTS ];
```

サンプルプログラム(3)

- 幾何形状オブジェクトの読み込み・初期化

```
void  LoadObjects()
{
    // オブジェクトの読み込み
    object[ 0 ] = LoadObj( "Car.obj" );
    ScaleObj( object[ 0 ], 5.0f,
        &object_size[ 0 ].x, &object_size[ 0 ].y, &object_size[ 0 ].z );
    object[ 1 ] = LoadObj( "Pyramid.obj" );
    ScaleObj( object[ 1 ], 2.0f,
        &object_size[ 1 ].x, &object_size[ 1 ].y, &object_size[ 1 ].z );

    object_pos[ 0 ].x = 0.0f;
    object_pos[ 0 ].y = 2.0f;
    object_pos[ 0 ].z = 0.0f;
    object_ori[ 0 ] = 180.0f;
    ...
```

サンプルプログラム(4)

- 描画処理(オブジェクト＋影の描画)

```
//  画面描画時に呼ばれるコールバック関数
void  DisplayCallback()
{
    // 画面をクリア

    // 変換行列(カメラ座標系からワールド座標系への変換行列)を設定

    // 光源位置を設定
    float  light0_position[] = { light_pos.x, light_pos.y, light_pos.z, 1.0 };
    glLightfv( GL_LIGHT0, GL_POSITION, light0_position );

    // 格子模様の床を描画

    // それぞれの幾何形状モデル＋影を描画
    for ( int i=0; i<NUM_OBJECTS; i++ )
    {
```

サンプルプログラム(5)

- 描画処理(オブジェクト＋影の描画)

```
// モデル座標系からワールド座標系への変換行列を計算
float  matrix[ 16 ];
glPushMatrix();
glLoadIdentity();
glTranslatef( object_pos[ i ].x, object_pos[ i ].y, object_pos[ i ].z );
glRotatef( object_ori[ i ], 0.0f, 1.0f, 0.0f );
glGetFloatv( GL_MODELVIEW_MATRIX, matrix );
glPopMatrix();

// 物体を描画
glMultMatrixf( matrix );
RenderObj( object[ i ] );

// 影を描画
// 現在の影の描画方法に応じて処理を呼び出し
...
```

影の描画に必要

単位行列で初期化

位置と水平向きにもとづいて変換行列を設定

変換行列を取得

ワールド座標系からカメラ座標系への変換行列が設定されている状態で、モデル座標系からワールド座標系への変換行列を右からかける

引数として matrix を渡す

サンプルプログラム(6)

- テクスチャマッピングによる影の描画
- ポリゴン投影による影の描画
- (シャドウ・ヴォリュームによる影の描画)

```
//  テクスチャマッピングによる影の描画(位置・向き、大きさ、高さを指定)
void  RenderTextureShadow(
        float obj_matrix[ 16 ], float size_x, float size_z, float shadow_y )

//  ポリゴン投影による影の描画
void  RenderProjectionShadow(
       Obj * obj, float obj_matrix[ 16 ], Vector & light_dir,
       float color_r, float color_g, float color_b, float color_a )
```

各関数の引数の定義は後程説明

今日の内容

- 影の描画のプログラム
- テクスチャマッピングによる影の描画
- 平面へのポリゴン投影による影の描画
- シャドウ・ヴォリューム
- シャドウ・マッピング
- 高度な影の描画技術
- 高度な描画技術
- レポート課題

テクスチャマッピングによる影の描画

テクスチャマッピングによる影の描画

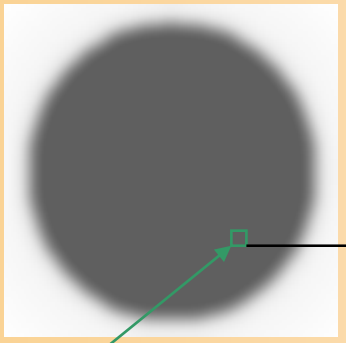
- 適当な影の画像を用意
- 物体の下に影の画像をテクスチャマッピング
  - 単純に貼りつけるとおかしくなるので、ブレンディング（半透明描画）を行いながら貼り付ける

影の画像

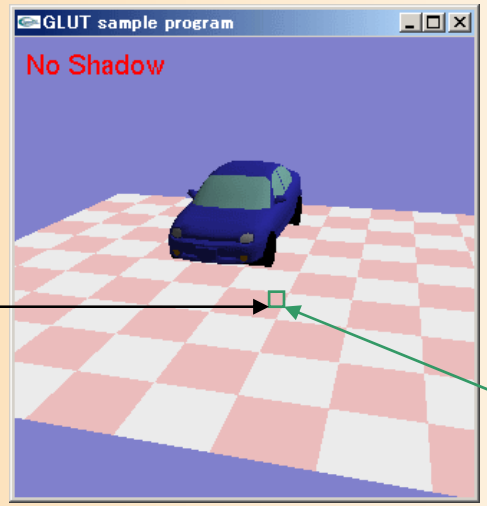
ブレンディングの方法

- glEnable(GL_BLEND)
- glBlendFunc(Fsrc, Fdest)
  - 描画色（この例ではテクスチャ）と画面のもとのピクセル色をどのように混ぜ合わせるかを設定

$$C = C_{src} * F_{src} + C_{dest} * F_{dest}$$



?



C_{src}（R, G, B, A）

C_{dest}（R, G, B, A）

ブレンディングの方法

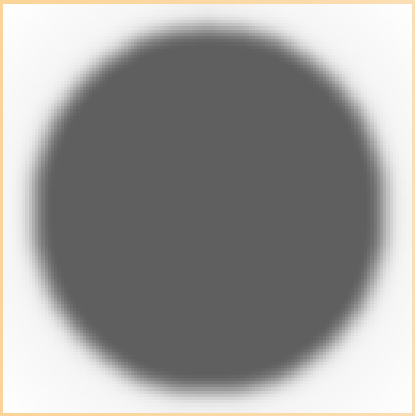
- glBlendFunc(Fsrc, Fdest) の引数の種類
  - GL_ZERO
  - GL_ONE
  - GL_DEST_COLOR
  - GL_SRC_COLOR
  - GL_ONE_MINUS_DEST_COLOR
  - GL_ONE_MINUS_SRC_COLOR

  色を係数として使用
  - GL_SRC_ALPHA
  - GL_DEST_ASPHA
  - GL_ONE_MINUS_SRC_ALPHA
  - GL_ONE_MINUS_DEST_ALPHA
  - GL_SATURATE

  アルファ値を係数として使用

影テクスチャの貼り付け(1)

- 方法1:アルファプレーンを使用
  - はりつける影の部分を指定してアルファプレーンを作成しておく
    - グレースケールBMPとして用意して読み込んでも良いし、テクスチャ画像から自動的に生成しても良い



カラープレーン（R, G, B）

＋

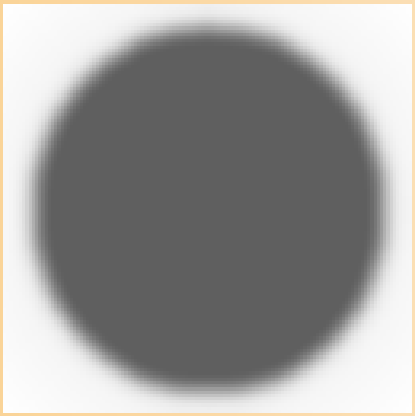
アルファプレーン（A）

どの程度テクスチャを画面に混ぜ合わせるかを表している
0 の範囲は全く描画されない

影テクスチャの貼り付け(2)

- 方法1:アルファプレーンを使用(続き)
  - テクスチャ側のアルファ値を使ってブレンド

```
glBlendFunc( GL_SRC_ALPHA,
             GL_ONE_MINUS_SRC_ALPHA );
```

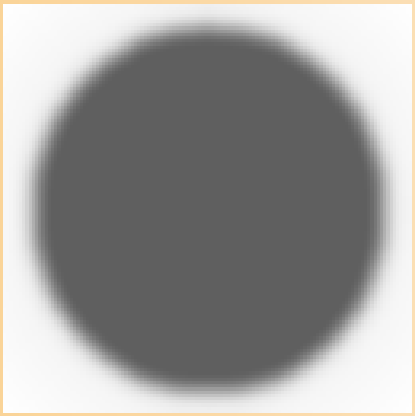


+

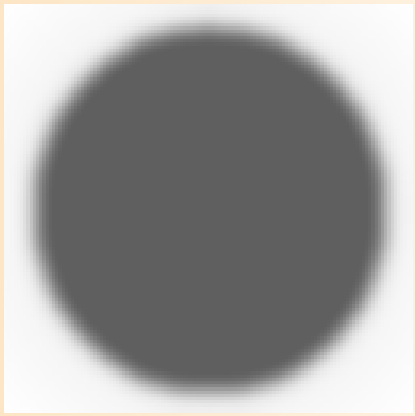
カラープレーン（R, G, B）　　アルファプレーン（A）

影テクスチャの貼り付け(3)

- 方法1:アルファプレーンを使用(続き)
  - 下図のようなアルファプレーンを使用するときは、2つの引数を入れ替える
    glBlendFunc(GL_ONE_MINUS_SRC_ALPHA,
    GL_SRC_ALPHA);



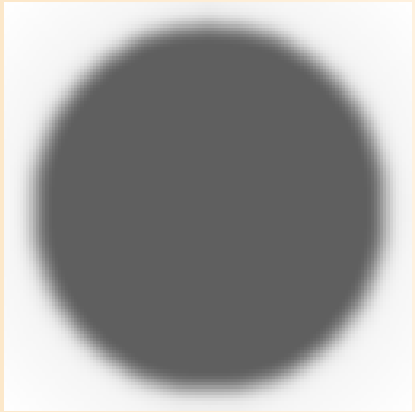
＋



カラープレーン（R, G, B）　　アルファプレーン（A）

影テクスチャの貼り付け(4)

- 方法2:カラープレーンのみを使用
  - 今回は、テクスチャの色によってブレンド比率が決まるので、アルファプレーンを使わなくて済む
  - テクスチャ側のカラー値を使ってブレンド
    glBlendFunc(GL_ONE_MINUS_SRC_COLOR, GL_SRC_COLOR);



カラープレーン（R, G, B）

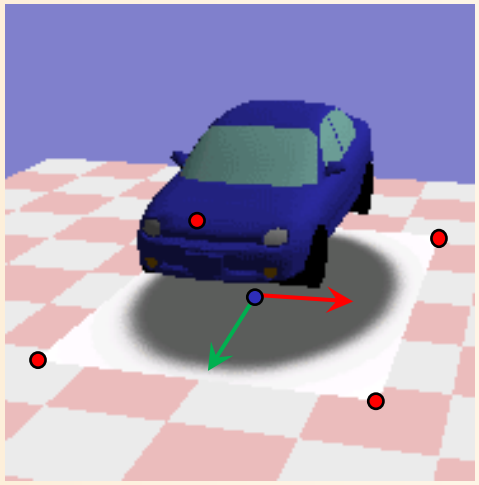
参考:テクスチャマッピングの手順

1. テクスチャ画像の読み込み
2. テクスチャ画像を登録
3. テクスチャマッピングのパラメタを設定
4. テクスチャ画像の適用方法を設定
5. テクスチャマッピングを用いてポリゴンを描画
   - テクスチャマッピングを有効に設定
   - 各頂点ごとにテクスチャ座標(u,v)を指定

※ 詳細は参考書・テキストを参照

影画像の描画処理の手順

1. テクスチャ画像の読み込み・設定
2. テクスチャ画像の描画位置（四角形ポリゴンの四隅の位置）を計算
3. テクスチャマッピングの設定
4. テクスチャ画像の描画
   （四角形ポリゴンを描画）



描画処理の作成(1)

```
// テクスチャマッピングによる影の描画
void  RenderTextureShadow( float obj_matrix[ 16 ],
              float size_x, float size_z, float shadow_y )
{
    // テクスチャ画像の読み込みと設定
    // 最初に一度だけ行われる
    if ( shadow_texture == 0 )
    {
        if ( ! LoadShadowTexture() )
            return;
    }

    ...
```

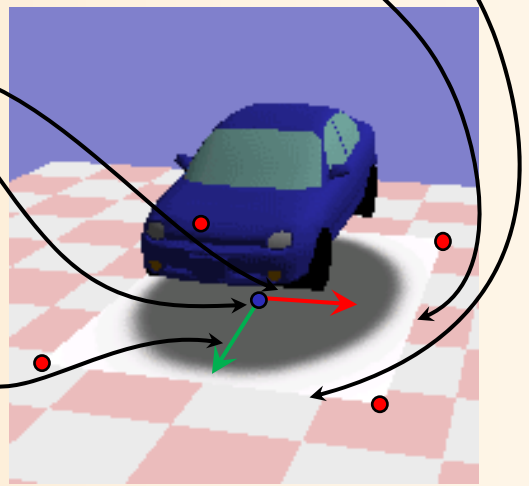
影を描画する物体の水平位置・向き、
影の前後・左右方向の大きさ、
影の高さ、を引数として受け取る

テクスチャ画像の読み込みと設定
詳しい処理はサンプルプログラムを参照

描画処理の作成(2)

- テクスチャ画像の描画(四隅)位置の計算
- 入力情報
  - オブジェクトの位置・向きを表す変換行列　$\mathbf{M}$
  - 影の左右・前後方向の大きさ　$size\_x, size\_z$
- 計算方法
  - 変換行列から
    水平方向の
    位置・X軸・Z軸
    を取得 → 四隅の位置の計算

$$\mathbf{M} = \begin{pmatrix} R_{11} & R_{12} & R_{13} & x \\ R_{21} & R_{22} & R_{23} & y \\ R_{31} & R_{32} & R_{33} & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

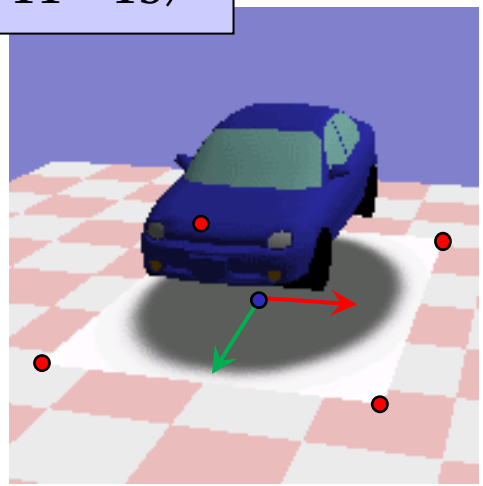


描画処理の作成(3)

```
// 影のテクスチャ画像を描画する四隅の水平位置＋高さ
float   x0, z0, x1, z1, x2, z2, x3, z3, y;

// オブジェクトの水平方向の中心位置・x軸方向・z軸方向を取得
float  center_x, center_z, x_axis_x, x_axis_z, z_axis_x, z_axis_z;
center_x = obj_matrix[ 12 ];
center_z = ？
・・・

// テクスチャ画像を描画する四隅の水平置を計算
x0 = center_x + (0.5 * size_x) * x_axis_x
              + (0.5 * size_z) * z_axis_x;
z0 = ？
・・・

// 高さは引数として渡された値をそのまま使用
y = shadow_y;
```

中心位置のx座標・z座標、
X軸のx座標成分・z座標成分、
Z軸のx座標成分・z座標成分

$$\begin{pmatrix} 0 & 4 & 8 & 12 \\ 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \end{pmatrix}$$



描画処理の作成(4)

```
// 現在の描画設定を取得(描画終了後に元の設定に戻すため)
...

// 描画オプションの設定
glDisable( GL_LIGHTING );  // ライティングは無効に設定
glEnable( GL_BLEND );  // ブレンディングを有効に設定
glEnable( GL_TEXTURE_2D );  // テクスチャマッピング

// テクスチャマッピングの設定
glBindTexture( GL_TEXTURE_2D, shadow_texture );
glTexEnvf( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL );

// ブレンディングの設定(方法2を使用)
glBlendFunc(                    ?                    );
```

アルファプレーンは設定されていないため、
カラープレーンのみを用いる方法を使用

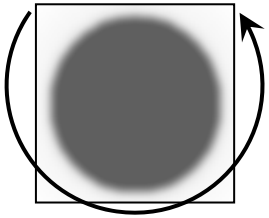
描画処理の作成(5)

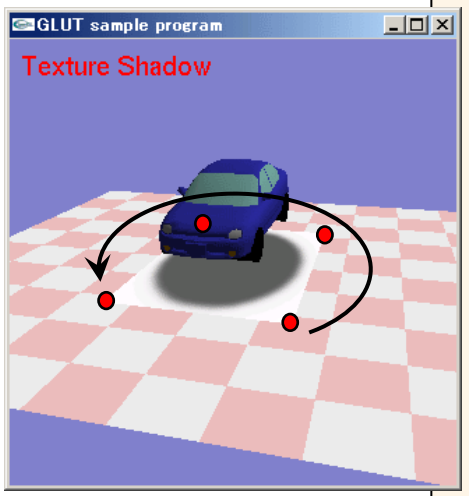
```
// 影テクスチャの描画(四角形のポリゴンを描画)
glBegin( GL_POLYGON );
    glNormal3f( 0.0, 1.0, 0.0 );
    glTexCoord2f( 1.0, 0.0 );
    glVertex3f( x0, y, z0 );
    ・・・
```

どの点から始めても構わないので、半時計周りに、四隅のテクスチャ座標・位置を指定

？

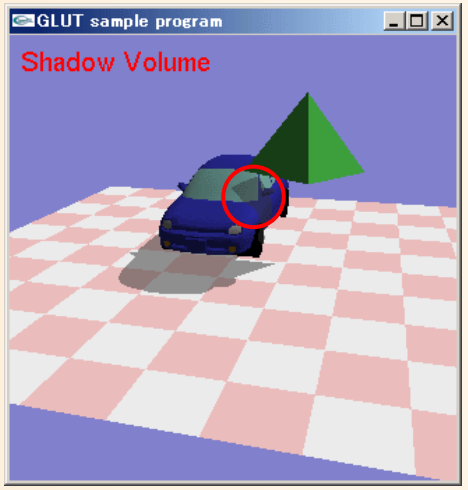
```
glEnd();

// 描画設定を復元
・・・
```





テクスチャマッピングによる影の描画の問題点

- 影の形が単純
- 水平面にしか影を投影できない
  - 他の物体や自分自身への影の投影はできない
- 物体同士が近くにあるときに、影テクスチャ同士が重なるとおかしくなる
  - ステンシルバッファを使った解決方法を次で説明



ポリゴン投影による影の描画

ポリゴン投影による影の描画

- 物体を構成する各ポリゴンを、地面に投影して、灰色（半透明）で描画
  - 物体の形状を反映した影を描画できる
  - 単純計算で、２倍の量のポリゴンを描画する必要がある
    - テクスチャマッピングによる影の描画と比べると、描画処理に時間がかかる

地面への投影(1)

- 視野変換行列の計算に、投影行列を追加
  - モデルからワールドへの変換行列 **M**
  - ワールドからカメラへの変換行列 **C**
  - ワールド座標系での地面への投影行列 P

    影を描画するときの視野変換行列 ＝ **C P M**

- 単純な投影行列 P
  - 真下に投影
    - y 座標の値を常に 0 にする

$$\mathbf{P} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

地面への投影(2)

- 任意の方向への投影
  - 光源の方向を（$light\_x, light\_y, light\_z$）とする

$$\mathbf{P} = \begin{pmatrix} 1 & -light\_x / light\_y & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -light\_z / light\_y & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- 任意の方向+任意の平面への投影
  - 複雑になるが、同様に変換行列を計算できる
  - 赤本を参照

地面以外への投影

- 各平面ごとにポリゴンを投影して描画すれば、地面以外の影も表現できる
  - ただし、影が平面からはみ出る場合は、切り取りのための処理が必要
  - クリッププレーンを追加すれば、OpenGLが処理してくれる

投影した幾何形状の影の描画

- 投影のための変換行列を設定した状態で、物体の幾何形状モデルを描画
  - 3次元の頂点座標は自動的に地面に投影される
- 頂点の色を全て黒（灰色）・半透明で描画する
  - ブレンディングにより半透明で描画
- ライティング（光源処理）はオフにして描画する

描画処理の作成(1)

```
//  ポリゴン投影による影の描画
void  RenderProjectionShadow( const Obj * obj,
            const float obj_matrix[ 16 ], const Vector & light_dir,
            float color_r, float color_g, float color_b, float color_a )
{
    // 描画オプションの設定
    glDisable( GL_LIGHTING );  // ライティングは無効に設定
    glEnable( GL_BLEND );  // ブレンディングを有効に設定
    glEnable( GL_STENCIL_TEST );  // ステンシルバッファを使用するよう設定

    // ブレンディングの設定
    glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );

    // ステンシルバッファの設定
    ・・・(後から追加、後述)

    // 動作確認のための描画オプションの変更
    ・・・
```

物体の幾何形状モデル、
物体の位置・向き、
光源の方向、影の色

描画処理の作成(2)

```
// 現在の変換行列を一時保存
glPushMatrix();

// ポリゴンモデルを地面に投影して描画するための変換行列を設定
// ワールド→カメラ変換 × 地面への投影変換 × モデル→ワールド変換
// この時点で、ワールド→カメラ変換の変換行列が設定されているものとする

// 地面への投影行列を計算
float  mat[ 16 ];
mat[ 0 ] = [ ? ];
・・・
mat[ 15 ] = [ ? ];

// 変換行列を設定
glMultMatrixf( [ ? ] );
glMultMatrixf( [ ? ] );
```

$$\mathbf{P} = \begin{pmatrix} 1 & -light\_x / light\_y & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -light\_z / light\_y & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

描画処理の作成(3)

```
    ・・・

    // 変換行列の設定
    ・・・

    // 影の描画、幾何形状モデルを指定色で描画
    RenderObjUnicolor( obj, color_r, color_g, color_b, color_a );

    // 一時保存しておいた変換行列を復元
    glPopMatrix();

    // 描画設定を復元
    ・・・
}
```

描画処理の作成(4)

- 影の描画
  - 幾何形状モデルを指定色で描画
    - 幾何形状モデルが持っている色の情報は使用せずに、全ての頂点を指定された色で描画する
  - 通常の描画関数(RenderObj関数)を参考に作成

```
//
//  幾何形状モデル(Obj形状)の描画(固定色で描画)
//
void  RenderObjUnicolor( const Obj * obj,
            float color_r, float color_g, float color_b, float color_a )
{
    ?
}
```

ブレンディング使用時の問題

- ブレンディングなし
  - 影が塗りつぶされて不自然
- ブレンディングあり
  - 複数のポリゴンが重なる箇所が暗くなってしまう
  - Zバッファが有効になっていれば同じ位置にポリゴンは重ならないはずだが、微妙な誤差のため複数回描画される点が生じる
  - ステンシルバッファを用いて回避

ステンシルバッファ

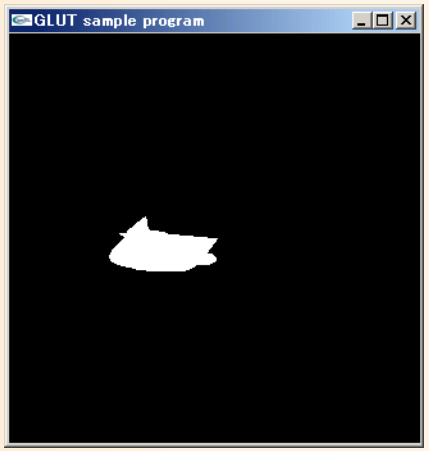
- 各ピクセルへの描画の可否を制御できる
  - Zバッファと同じく、画面と同サイズの領域を持つ
  - 各ピクセルには整数値を書き込むことができる
  - Zバッファと同じく、ある条件を満たすときだけ描画が行えるように、条件を設定できる

フレームバッファ(R, G, B, A)

＋

Zバッファ

＋

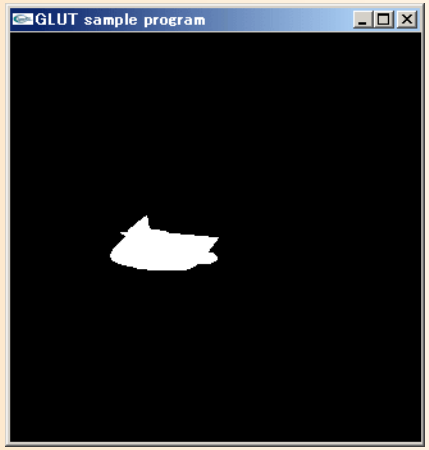


ステンシルバッファ

ステンシルバッファを使った描画

- 影の重ね描きを防ぐためのフラグとして使う
  - 影のポリゴンの各ピクセルを描画するときに、ステンシルバッファに値を書き込む
  - 既にステンシルバッファに値が書き込まれていれば、そのピクセルには描画しない

カラーバッファ(R, G, B, A)



ステンシルバッファ

ステンシルバッファの初期化

- ステンシルバッファの初期化
  - 通常はステンシルバッファを持たない
  - 初期化時に指定する必要がある
  - 通常、ピクセル当たり 1bit～8bit 程度を使用（色情報の一部を利用することもある）
- GLUTでのステンシルバッファの利用方法
  - glutInit() の引数に GLUT_STENCIL を指定
  - グラフィックカード・ドライバ・画面モードによっては、必ずしも成功するとは限らない

ステンシルバッファの利用

- ステンシルバッファのクリア
  - glClear(GL_STENCIL_BUFFER_BIT);
- ステンシルテストの有効化
  - glEnable(GL_STENCIL_TEST);
- ステンシルテストの設定
  - 各ピクセルのステンシルバッファの値にもとづいて描画の可否を判定
- ステンシルバッファへの書き込みの設定

ステンシルテストの設定

- glStencilFunc(func, ref, mask)
  - func には比較関数の種類を指定
    - GL_EQUAL, GL_NOTEQUAL, GL_LESS, GL_GREATER, GL_LEEQUAL, GL_GEEQUAL, GL_NEVER, GL_ALWAYS
    - 現在のステンシルバッファの値と ref の値を比較して、条件を満たすときにのみ書き込みを行う
  - 判定を行う前に、ステンシル値に mask との論理積を適用（一部のビットのみを参照できる）
  - 例： glStencilFunc(GL_NOTEQUAL, 1, 1);
    - ステンシル値が 1 以外の場合のみ、描き込み

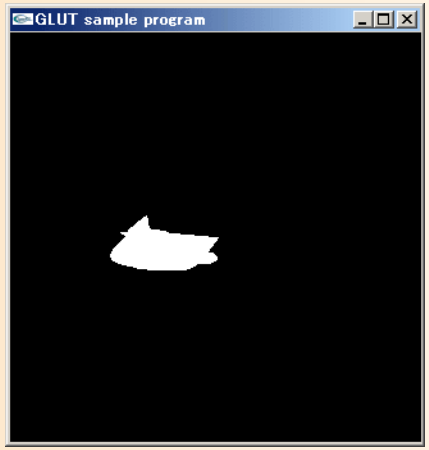
ステンシルバッファへの書き込み

- glStencilOp(fail, zfail, zpass)
  - それぞれ、ステンシルテストに失敗（fail）、ステンシルテストは通ったがZテストに失敗（zfail）、どちらも成功してピクセルを更新（zpass）したときに、ステンシルバッファをどうするかを設定
  - GL_KEEP, GL_ZERO, GL_REPLACE, GL_INCR, GL_DECR, GL_INVERT
    - GL_REPLACE では、参照値 ref を書き込む
  - 例：glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);
    - ピクセル描画時に、ステンシル値に ref を代入

ステンシルバッファを使った描画

- 影の重ね描きを防ぐためのフラグとして使う
  - 既に影が描かれたピクセルのステンシル値を 1 とする
  - glStencilFunc(GL_NOTEQUAL, 1, 1);
    - 既にステンシル値が 1 のピクセルには描画を行わない
  - glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);
    - ピクセルに描き込むときにステンシル値を 1 に設定

カラーバッファ（R, G, B, A）



ステンシルバッファ

描画処理の変更

```
// 描画オプションの設定
glDisable( GL_LIGHTING );  // ライティングは無効に設定
glEnable( GL_BLEND );  // ブレンディングを有効に設定
glEnable( GL_STENCIL_TEST );  // ステンシルバッファを使用するよう設定

// ブレンディングの設定
glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );

// ステンシルバッファの設定
glStencilFunc(  ？  );
glStencilOp(    ？  );

// 変換行列の設定
・・・

// 幾何形状モデルを描画（指定色で描画）
RenderObjShadow( obj, color_r, color_g, color_b, color_a );
```

実行結果

- 影のポリゴンが重なることなく描画される
- 複数オブジェクトの影の重なりにも対応可能
- テクスチャマッピングによる影の描画と同じく、基本的に地面にしか投影できないという問題がある

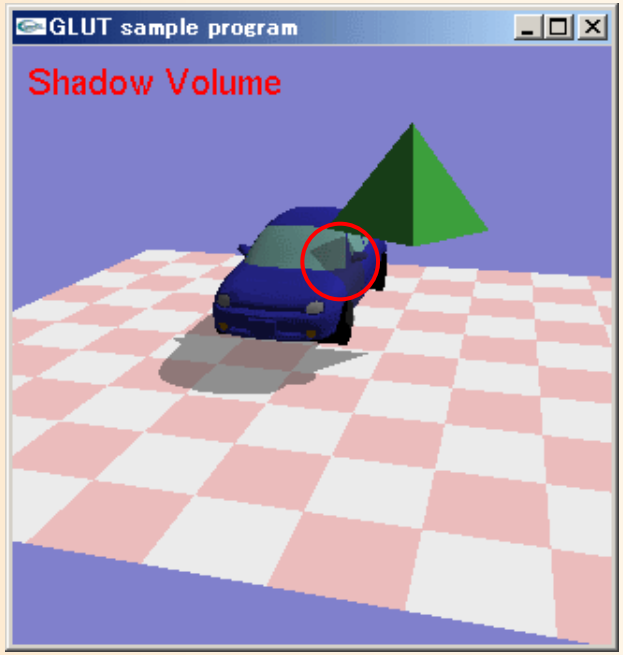
今日の内容

- 影の描画のプログラム
- テクスチャマッピングによる影の描画
- 平面へのポリゴン投影による影の描画
- シャドウ・ヴォリューム
- シャドウ・マッピング
- 高度な影の描画技術
- 高度な描画技術
- レポート課題

シャドウ・ヴォリューム

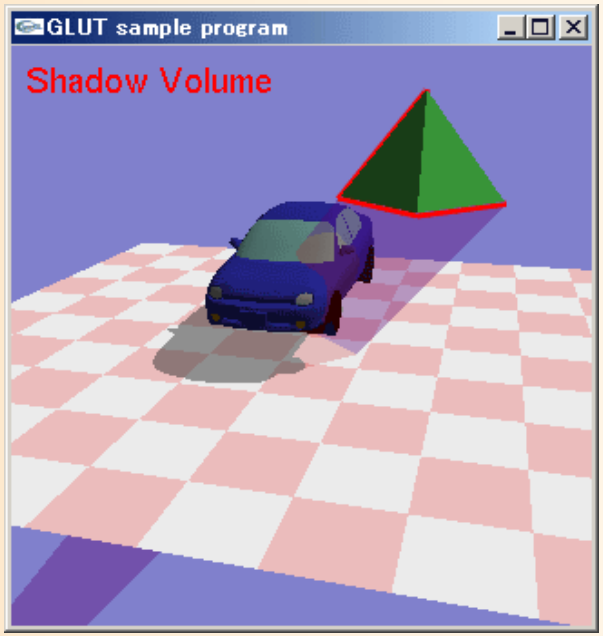
シャドウ・ヴォリューム

- 他の物体や自分自身に投影される影も実現
- 影になる空間領域（シャドウ・ヴォリューム）を求める
  - ステンシルバッファを利用



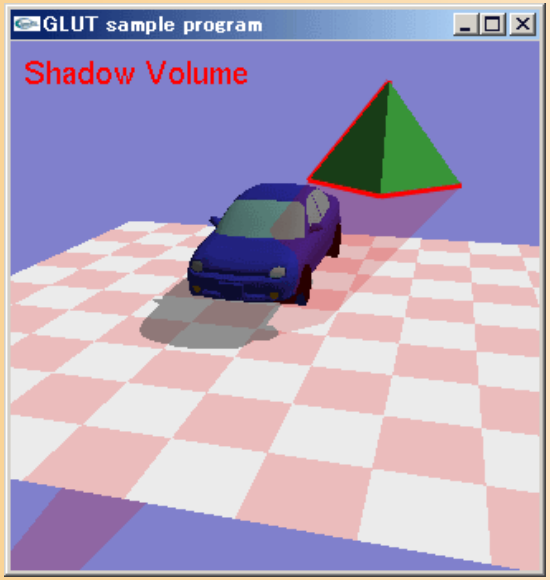
描画手順(1)

- 光源から見て物体の輪郭になる辺を求める
- 輪郭になる辺を光の伸びる方向に拡張し、シャドウ・ヴォリュームを作成

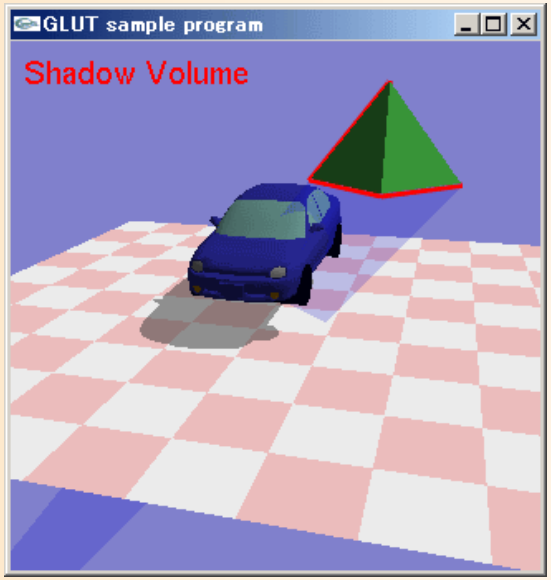


描画手順(2)

- シャドウ・ヴォリュームの表の面の描画処理を行い、ステンシルバッファの値を加算
  - 実際の描画は行わない
- 裏の面も同様に描画処理を行い、減算



−



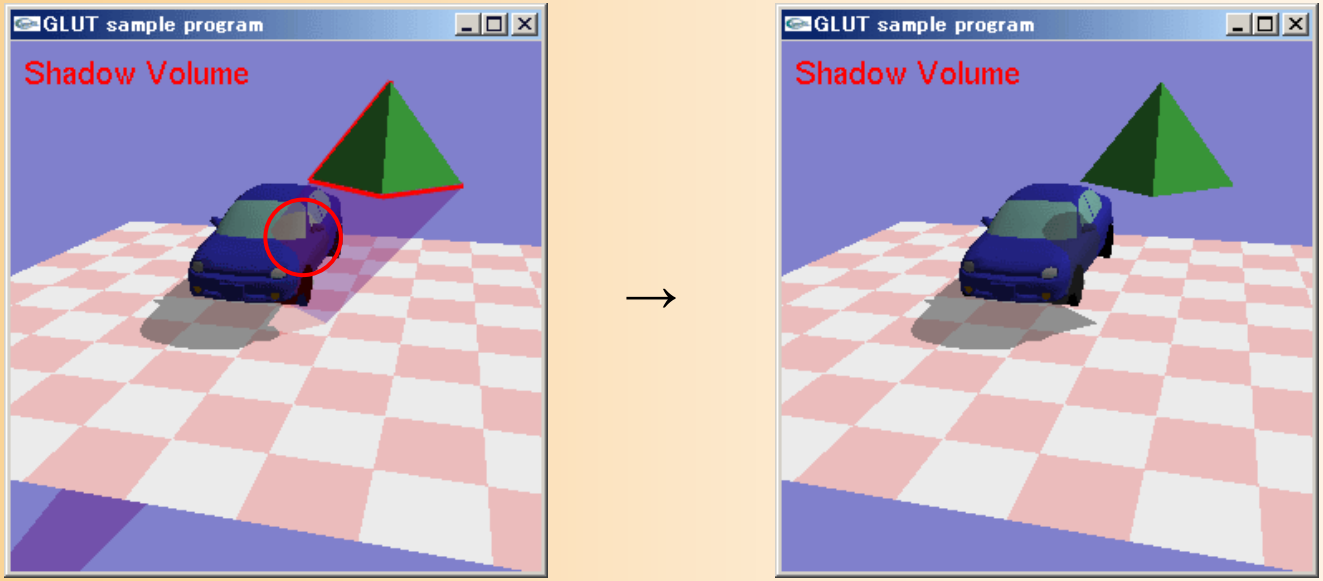
=

表面

裏面

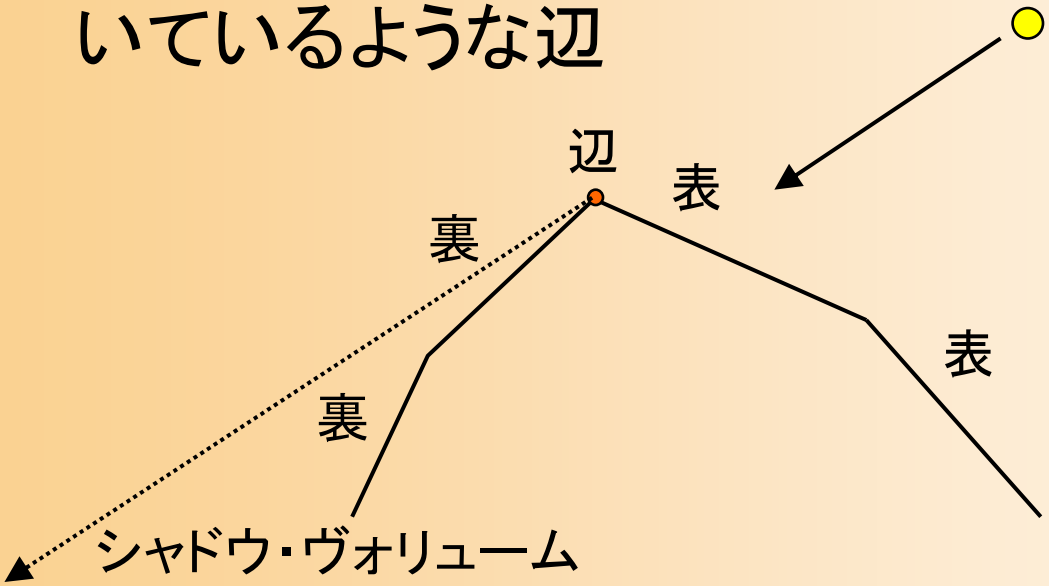
描画手順(3)

- Zバッファ法による描画の結果、影の領域のみステンシルバッファの値が1以上になる
  - 表面よりも後ろで、裏面よりも前にある領域
- その領域にのみアルファブレンドを適用



輪郭辺の計算(1)

- 輪郭辺の定義
  - 光源から見た物体の輪郭辺
  - 辺の両側の面のうち、片側の面が光源方向を向いており、もう片側の面が光源と反対方向を向いているような辺

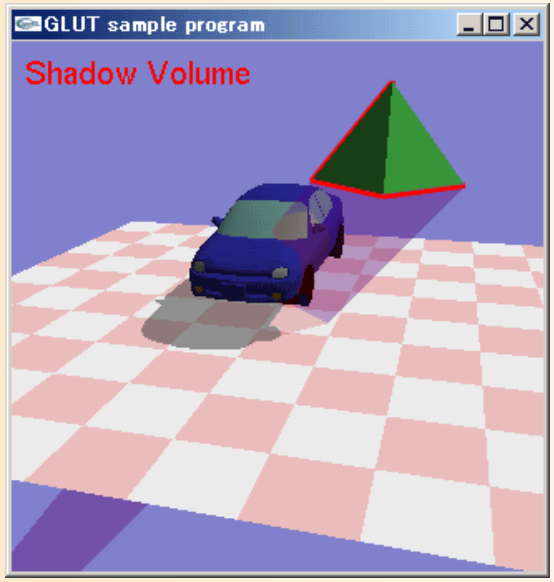


輪郭辺の計算(2)

- 各辺と各面の対応関係を前計算しておく
  - 各辺に通し番号をつけて、各辺の両側の面の番号を記録しておく
  - (通常のポリゴンモデルは辺の情報は持たない)
- 物体が移動する度に、各面が光源方向を向いているかどうかを判定して記録
- 上記の2つの情報をもとに、各辺が輪郭辺かどうかを判定して記録
  - このとき、辺のどちら側が表かを記録しておく

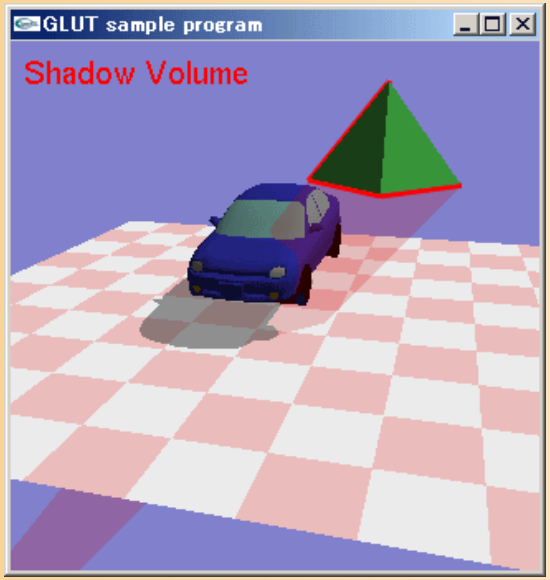
シャドウ・ヴォリュームの生成

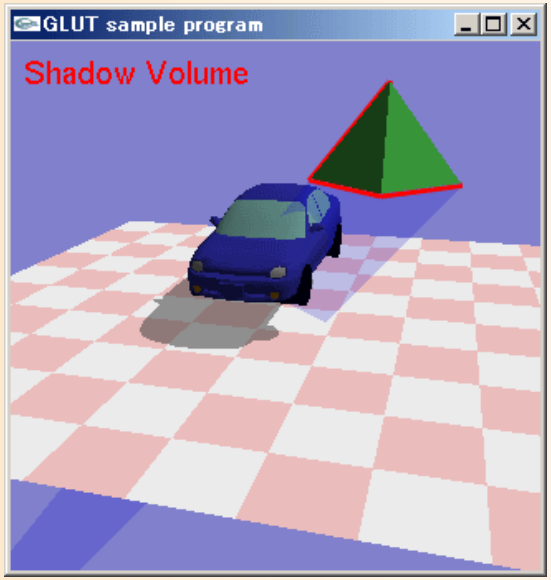
- 輪郭辺を光源と反対方向に延長
  - 各辺から四角面を生成
  - 四角面が表向きになるように、頂点の順番を合わせる



シャドウ・ヴォリュームの描画

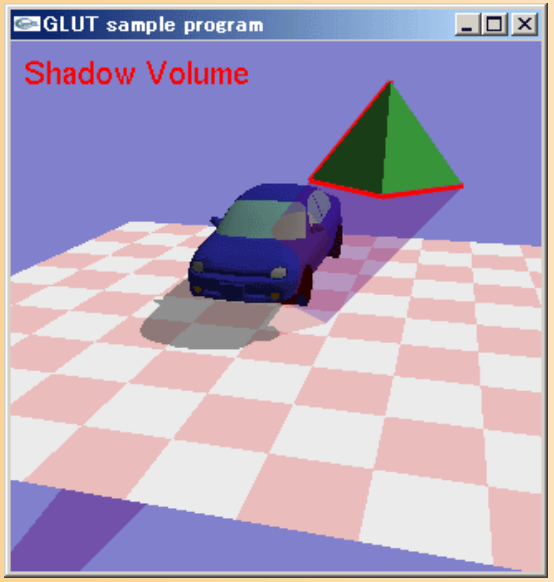
- 背面除去の機能を利用する
  - 表の面だけを描画（ステンシルバッファ加算）
  - 裏の面だけを描画（ステンシルバッファ減算）
  - glCullFace(GL_FRONT or GL_BACK);





影の領域を暗くする

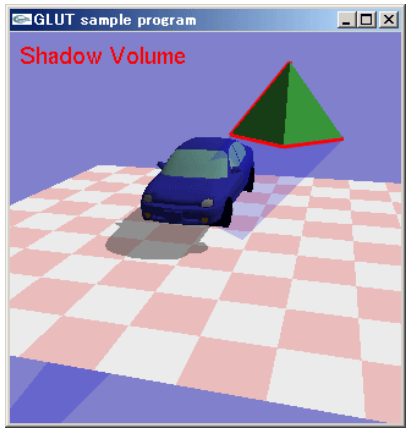
- 画面全体に、黒（灰色）の四角形を、半透明（アルファブレンディング）で描画
  - 平行投影を行うように設定（以前のテキスト描画と同様）
  - ステンシルテストを有効にして、画面全体に描画

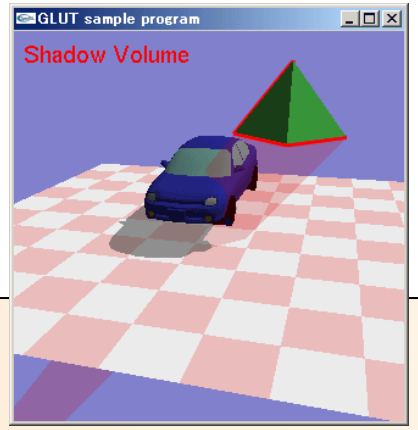


→

描画処理の例(1)

```
// 光源から見たときの物体の輪郭線を計算
vector< Vector >  contour_edges;  // 輪郭線を構成する各辺の頂点座標の配列
ComputeContourEdges( obj, contour_edges ); // 実装の詳細は省略

// シャドウ・ヴォリュームの前方の面を描画
glStencilFunc( GL_ALWAYS, 0, 0 );
glStencilOp( GL_KEEP, GL_KEEP, GL_INCR );
glCullFace( GL_BACK );
DrawVolume( contour_edges ); // 実装の詳細は省略

// シャドウ・ヴォリュームの後方の面を描画
glStencilFunc( GL_GREATER, 0, 0xff );
glStencilOp( GL_KEEP, GL_KEEP, GL_DEC );
glCullFace( GL_FRONT );
DrawVolume( contour_edges ); // 実装の詳細は省略
```





描画処理の例(2)

```
// 画面全体に描画するための射影行列を設定(演習資料の文字描画の解説を参照)

// ブレンディングの設定
glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );

// ステンシルバッファの設定
glStencilFunc( GL_GREATER, 0, 0xff );
glStencilOp( GL_KEEP, GL_KEEP, GL_KEEP );

// 画面全体を黒く描画
glBegin( GL_QUADS );
    glColor3f( 0.0f, 0.0f, 0.0f, 0.5f ); // α値で半透明度を指定
    glVertex3f( 0.0f, 0.0f, 0.0f );     glVertex3f( 0.0f, 1.0f, 0.0f );
    glVertex3f( 1.0f, 1.0f, 0.0f );     glVertex3f( 1.0f, 0.0f, 0.0f );
glEnd();

// 射影行列・描画設定を復元
```

問題点

- 複雑な物体ではうまくいかないときがある
  - 輪郭が曲面になっている場合などで、輪郭線が正しく判定されないことがある
- カメラがシャドウ・ヴォリュームの中に入ると、正しい画像が生成されない
- 処理時間がかかる
  - 輪郭辺の計算
    - ハードウェアでの実現が困難
  - シャドウ・ヴォリュームの描画
  - 画面全体への四角形の描画

輪郭辺の計算の応用

- 視線から見た輪郭を計算して描画することで、アニメ絵風の効果が出せる(トゥーン・レンダリング)

視線方向から見た輪郭を計算

シャドウ・マッピング

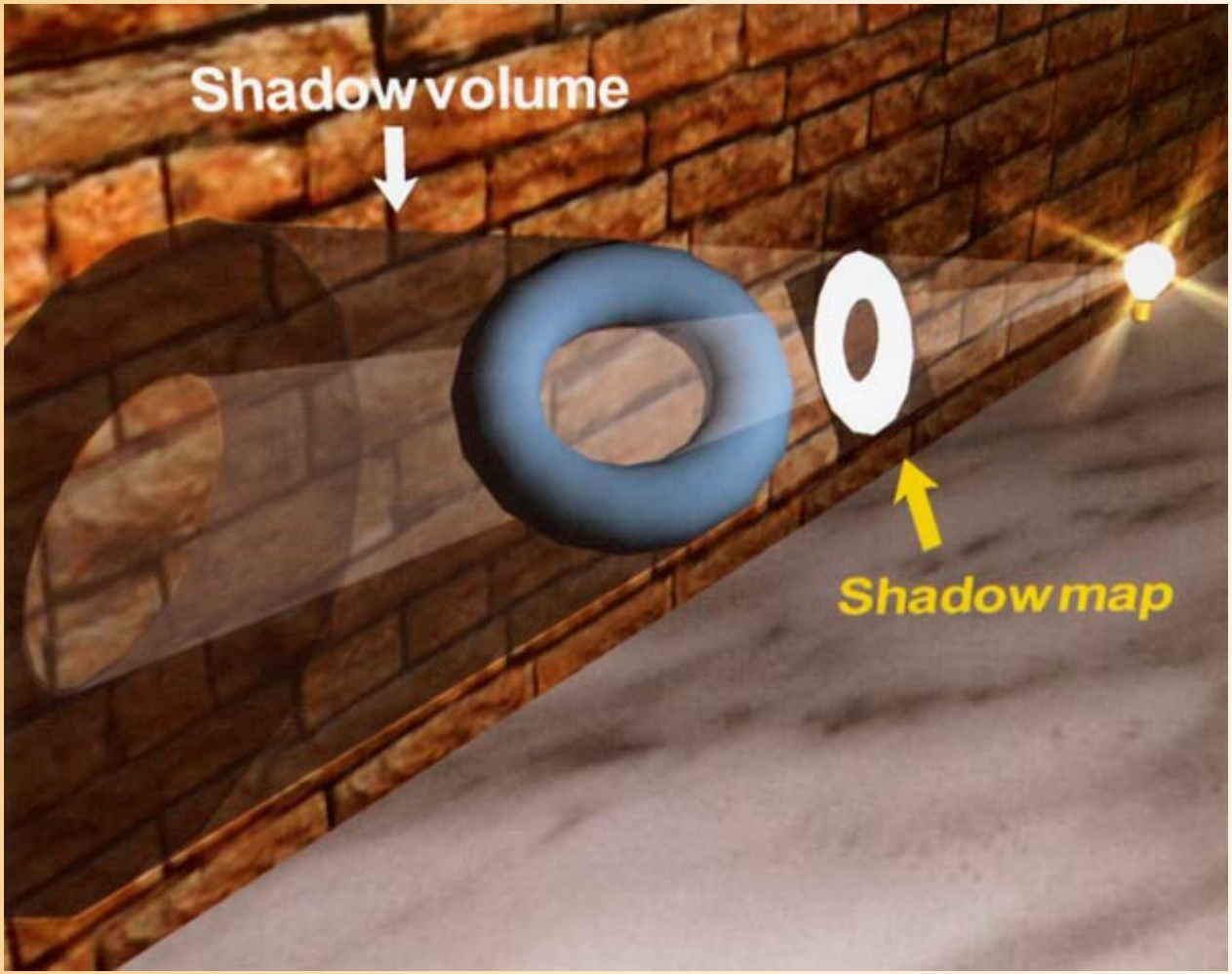
シャドウ・マッピングの概要

- マルチパス・レンダリング

1.まず光源から物体を見た画像をレンダリング
  - この結果をシャドウマップとする

2.物体が投影される面を描画するときに、シャドウマップをテクスチャマッピングする
  - 適切な位置に投影されるように、各頂点のテクスチャ座標を計算

シャドウ・マッピングの図解



[Game Programming GemsII, Gabor Nagy]

シャドウ・マッピングの詳細

- 詳しい実装方法については省略
  - 基本的にはこれまでに紹介した技術の組み合わせで実現できる
  - テクスチャへのレンダリングが必要になる
- シャドウ・マッピングの種類の違い
  - シャドウ・マッピング（奥行き値なし）
  - 深度マッピング（奥行き値あり）
- スキャンライン法と組み合わせて、オフラインでの高品質なレンダリングにも用いられる

高度な影の描画技術

セルフ・シャドウ

- 自分自身への影
- 実現方法
  - シャドウ・ヴォリューム を使えば、実現可能
  - シャドウ・マッピングで実現することは困難 （腕と胴体を別の物体として描画するなどの工夫が必要）

［SCE, ワンダと巨像 ］

参考資料：
3Dゲームファンのための「ワンダと巨像」グラフィックス講座
http://www.watch.impress.co.jp/game/docs/20051207/3dwa.htm

ソフト・シャドウ

- 輪郭がぼやけたような影
  - 現実世界では、ひとつの点光源ではないので、本来は影の輪郭はぼやける

[GPU GemsII, Yury Uralsky]

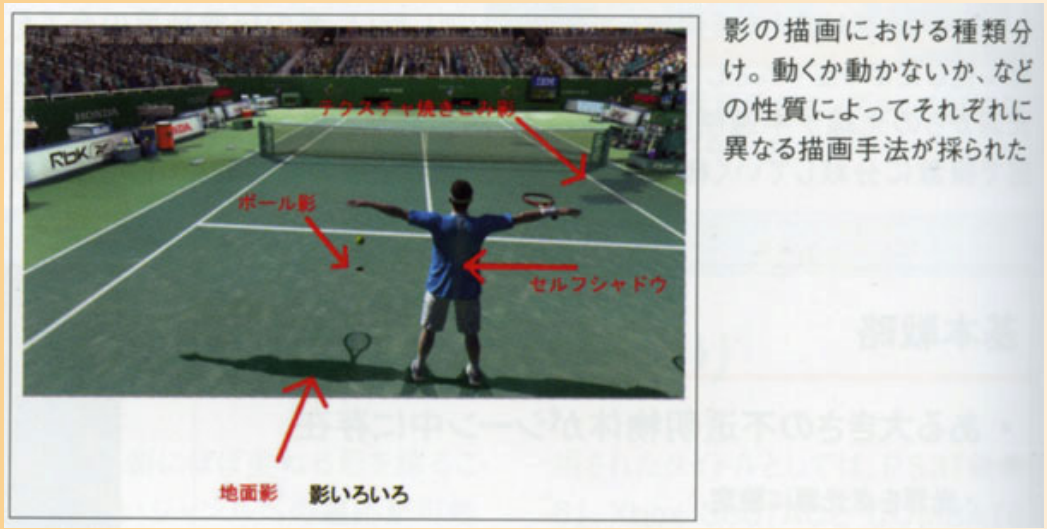
- 実現方法
  - シャドウ・ヴォリュームでは、実現は困難（光源を微妙にずらして複数回レンダリングなどすれば、時間はかかるが可能）
  - シャドウ・マッピングでは、シャドウマップをぼかしたりすることで、実現可能
  - ソフト・シャドウの描画に特化した手法もある

影の実現方法の比較

- シャドウ・ヴォリューム
  - シーンのポリゴン数に大きく影響を受ける
  - ある程度高いフィルレートが必要
  - 機能自体は、古いハードウェアでも実行可能
  - ソフトシャドウの実現は困難
- シャドウ・マップ
  - シーンのポリゴン数にはあまり影響は受けない
  - オフスクリーンレンダリングやマルチテクスチャに対応した環境が必要
  - セルフシャドウの実現は困難

影の描画方法の使い分け

- コンピュータゲームでの影の使い分けの例



影の描画における種類分け。動くか動かないか、などの性質によってそれぞれに異なる描画手法が採られた

［セガ, パワースマッシュ３ ］

参考資料:
CG WORLD 2007年12月号
「CEDEC 2007 技術トラック解説」

  - シャドウマップによるセルフシャドウ（人物）
  - 動く影（人物やボールから地面への影）は、ポリゴン投影（テニスコートは平面であることを利用）
  - 動かない影は地面のテクスチャに焼き込み

高度な描画技術

高度な描画技術

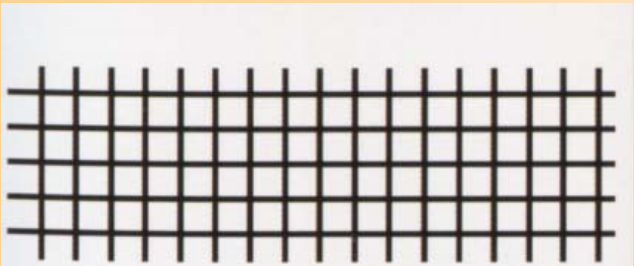
- 今回は影の描画のみを扱ったが、自然な画像を生成するための高度な描画技術は多く開発・利用されている
  - 本科目では扱わない
- 大域照明や物体表面の反射特性をより正確に実現するためのレンダリング技術など

高度な描画技術

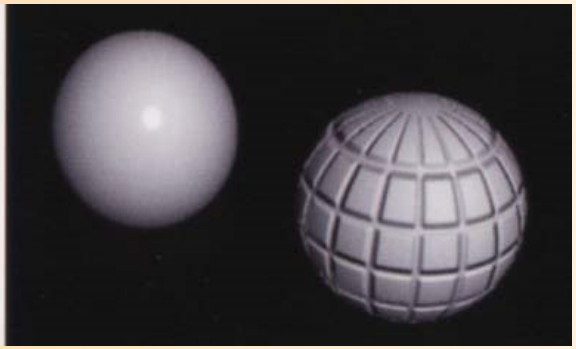
- バンプマッピング
  - 表面の凹凸を表す画像を適用し、法線を変化させることで、凹凸を表現
- 環境マッピング
  - 周囲の風景の映り込みを、半透明のテクスチャマッピングにより表現
- 大域照明（ラジオシティ、フォトンマップ）
  - 各面に当たる環境光をより正確に計算
  - 事前計算しておいたデータを適用

高度なマッピング（復習）

- 凹凸のマッピング（バンプマッピング）



基礎と応用 図5.9

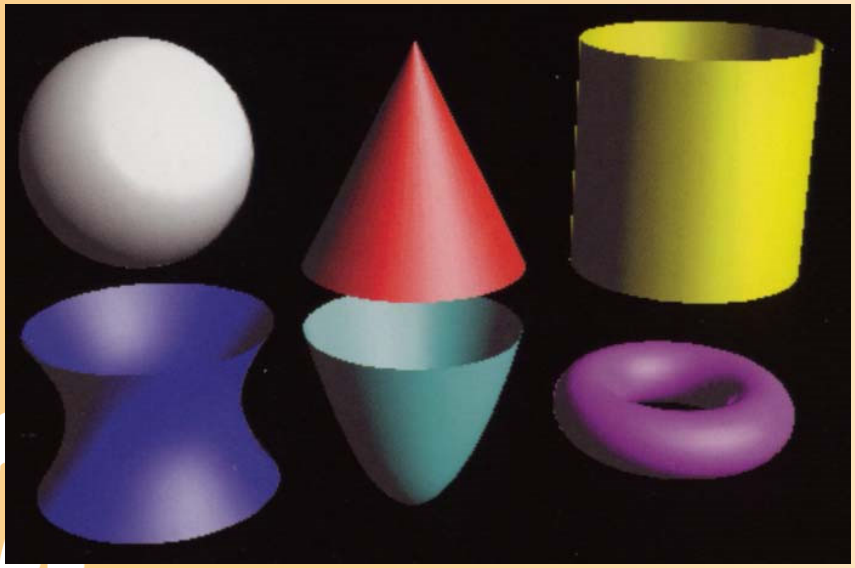


- 周囲の風景のマッピング（環境マッピング）

CG制作独習事典p.17

大域照明の効果の例(復習)

- 大域照明を考慮して描画することで、より写実的な画像を得ることができる



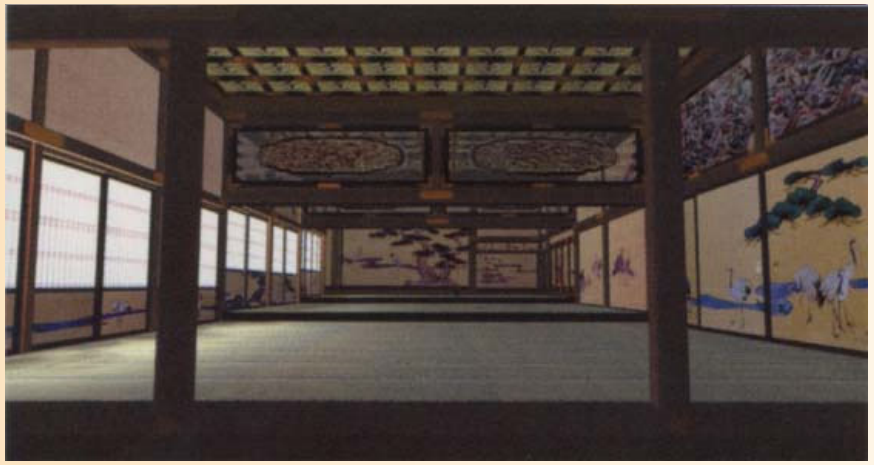
映り込み(大域照明)を考慮
基礎と応用 図8.9

環境光(大域照明)を考慮
基礎と応用 図9.1, 9.2

大域照明の効果の例(復習)

- 大域照明を考慮して描画することで、より写実的な画像を得ることができる

映り込み(大域照明)を考慮
基礎と応用 図8.9



環境光(大域照明)を考慮
基礎と応用 図9.1, 9.2

最近の高度な描画技術

- Precomputed Radiance Transfer（PRT）
  - 事前計算した輝度放射伝搬［Sloan 2002］
    - 物体の各頂点で、各方向から来た光によってどのように照らされるかを事前に計算しておく
    - 球面調和関数（Spherical Harmonics）で表現
  - 静的なシーン → 動的なシーンへの拡張
- Bi-directional Reflectance Distribution Function（BRDF）
  - 計測データにもとづき表面の反射特性を再現

まとめ

- 影の表現方法
  - テクスチャ
  - 平面へのポリゴン投影
  - シャドウ・ヴォリューム
  - シャドウ・マッピング
- OpenGLの高度な描画技術
  - アルファブレンディング
  - ステンシルバッファ
- 高度な影の描画技術
  - セルフ・シャドウ、ソフト・シャドウ

レポート課題

- 影の描画を実現するプログラムを作成せよ
  1. テクスチャマッピングによる影の描画
  2. ポリゴン投影による影の描画
  - サンプルプログラム（shadow_sample.cpp）をもとに作成したプログラムを提出
    - 他の変更なしのソースファイルやデータは、提出する必要はない
  - Moodleの本講義のコースから提出
  - 締切：Moodleの提出ページを参照

レポート課題 提出方法

Moodleから、以下の2つのファイルを提出

- 作成したプログラム（テキスト形式）
  - shadow_sample.cpp
- 変更箇所のみを書き出したレポート（PDF）
  - Moodle に公開している LaTeX のテンプレートをもとに、作成する
    - 前回のレポートと同様

レポート課題 演習問題

- レポート課題の提出に加えて、レポート課題の理解度を確認するための Moodle 演習問題にも解答する
  - 解答締切は、レポート提出と同じ
    - 締切までは解答を変更可、締切後に正解が表示
  - レポート課題のヒントにもなっているので、レポート課題で分からない箇所がえれば、演習問題の説明・選択肢を参考にして考えても良い
  - 本演習問題の点数は、演習課題の成績の一部として考慮する

レポート課題 発展

- より高度な技術に興味があれば、以下の処理を実現するような拡張が可能

- 頂点配列を使った幾何形状モデルの描画
  - 頂点配列の使用に適したデータ表現への変換
- シャドウ・ヴォリュームによる影の描画
- シャドウ・マップによる影の描画

次回予告

- キーフレームアニメーション
  - 行列・ベクトルを扱うプログラミング
  - 位置補間
    - 線形補間、Hermit曲線、Bézier曲線、B-Spline曲線
  - 向きの補間
    - オイラー角
    - 四元数と球面線形補間