コンピューターグラフィックスS

第6回　レンダリング・パイプライン

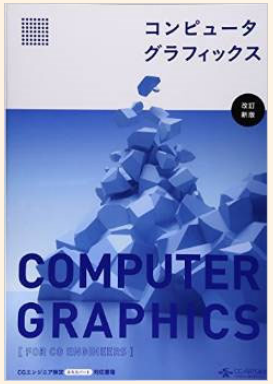
システム創成情報工学科　尾下 真樹

2019年度 Q2

今回の内容

- レンダリング・パイプライン
  - Zバッファ法によるポリゴン描画の仕組み
  - 座標変換とラスタライズ
- 座標変換
- ラスタライズ
- OpenGLでのレンダリング設定
- OpenGLでのポリゴン描画

教科書(参考書)

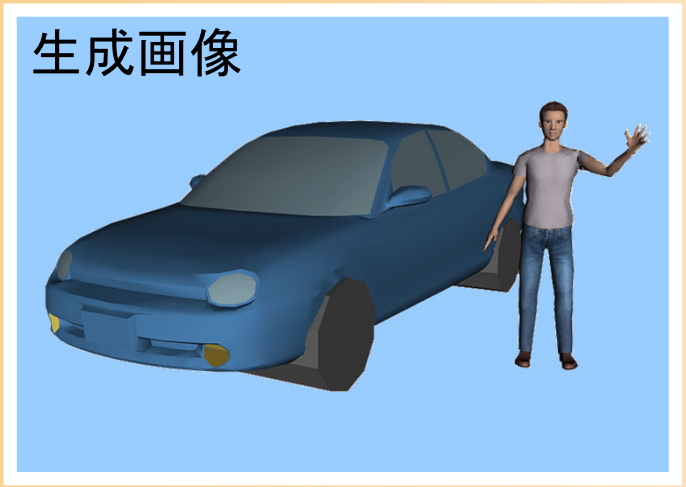


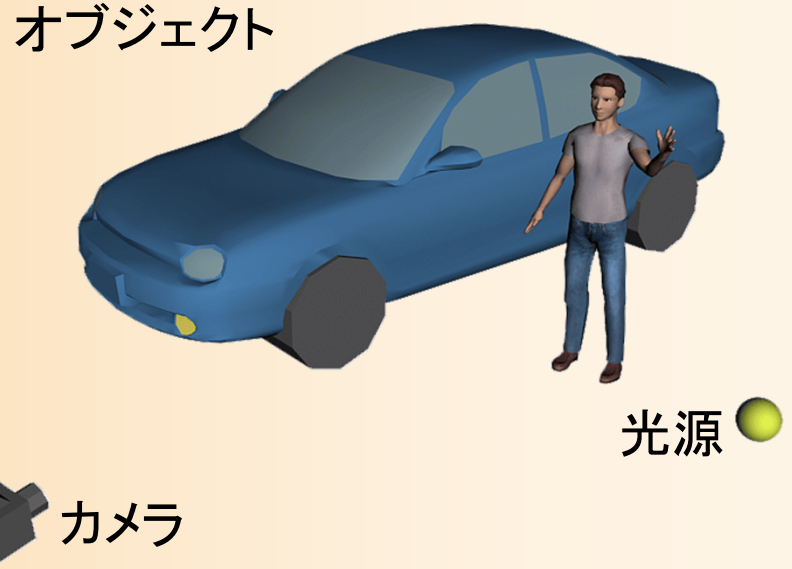
- 「コンピュータグラフィックス」
  CG-ARTS協会 編集・出版
  - 2章
- 「ビジュアル情報処理 ―CG・画像処理入門―」
  CG-ARTS協会 編集・出版
  - 1章(8～28ページ)
  - ごく簡単な説明のみ

前回の復習

レンダリング

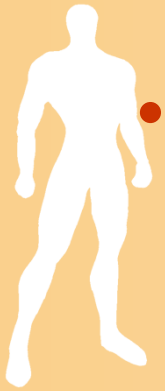
- レンダリング（Rendering）
  - カメラから見える画像を計算するための方法
  - 使用するレンダリングの方法によって、生成画像の品質、画像生成にかかる時間が決まる





レンダリングの予備知識

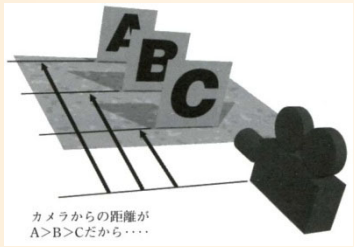
- 隠面消去
- カメラとビューポート
- 座標変換
- 面単位での描画
- 光のモデル
- 反射･透過･屈折の表現



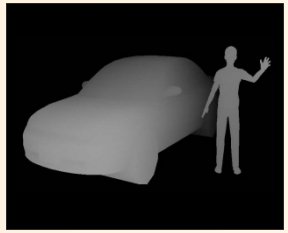
レンダリング手法

- Zソート法
- Zバッファ法
- スキャンライン法
- レイトレーシング法

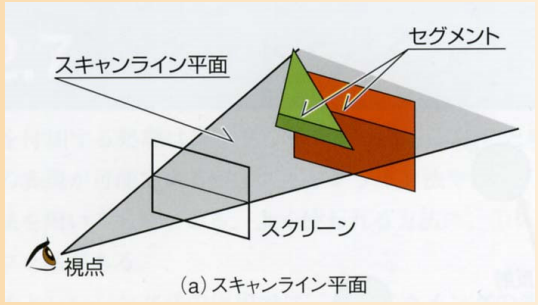
Zソート法



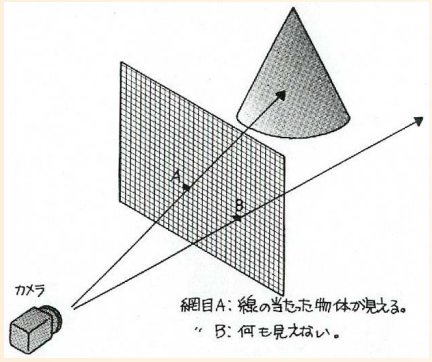
Zバッファ法



スキャンライン法

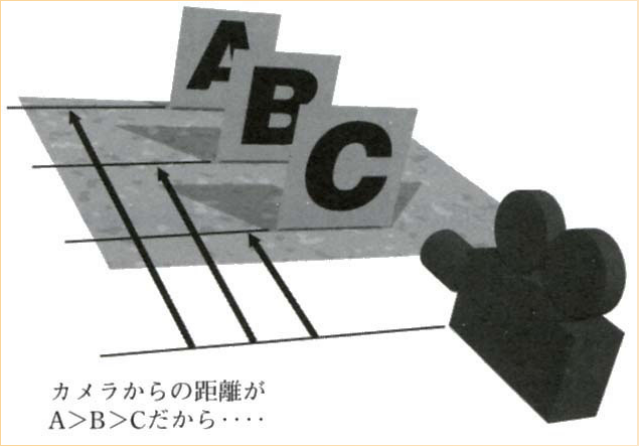


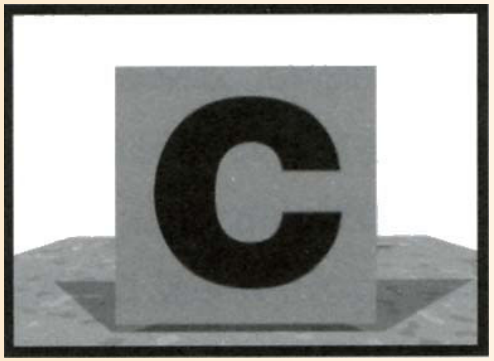
レイトレーシング法



Zソート法

- Zソート法（ペインタ・アルゴリズム）
  - 描画すべき面を奥にあるものから手前にあるものに順番にソート（整列）
  - 奥の面から手前の面へ順番に描画していく
  - 結果的に、奥の面は手前の面で隠れる

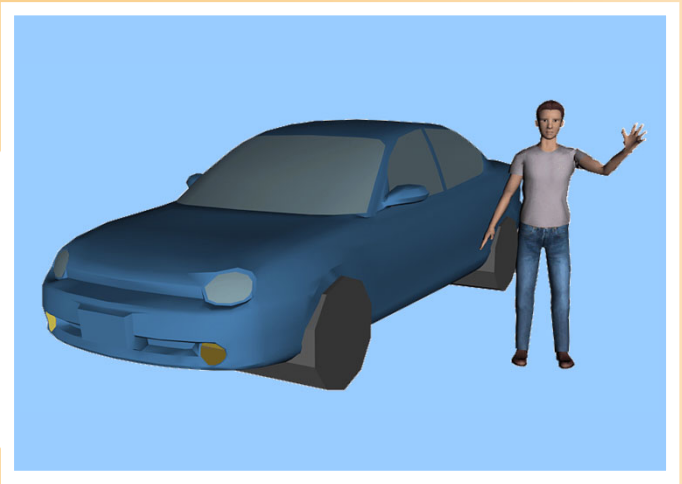


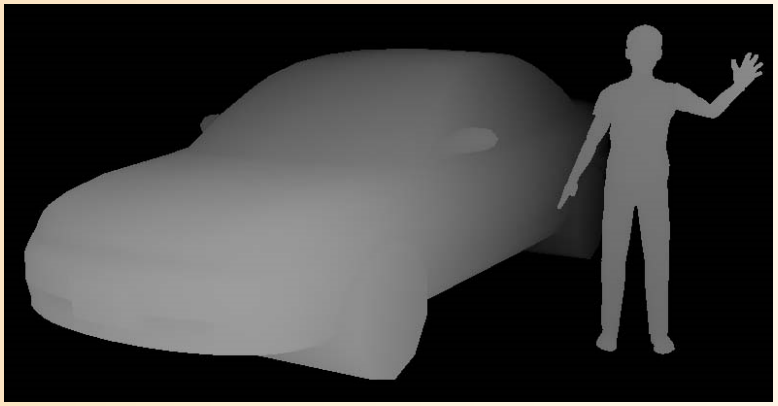


教科書 基礎知識 図2-30

Zバッファ法

- 画像とは別に、それぞれのピクセルの奥行き情報であるZバッファを持つ
  - Zバッファは画像とほぼ同じメモリサイズを使用
    - ピクセルあたり16ビット～32ビットを使用
  - ピクセル単位で処理するので交差も処理できる

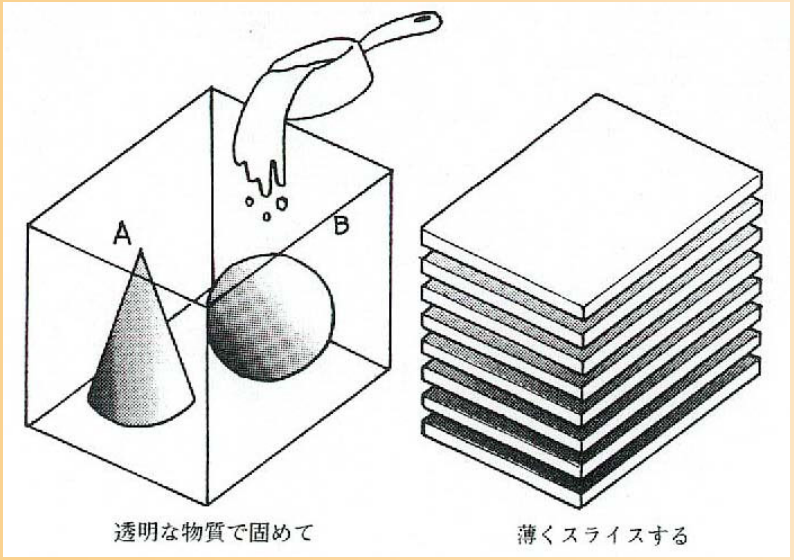


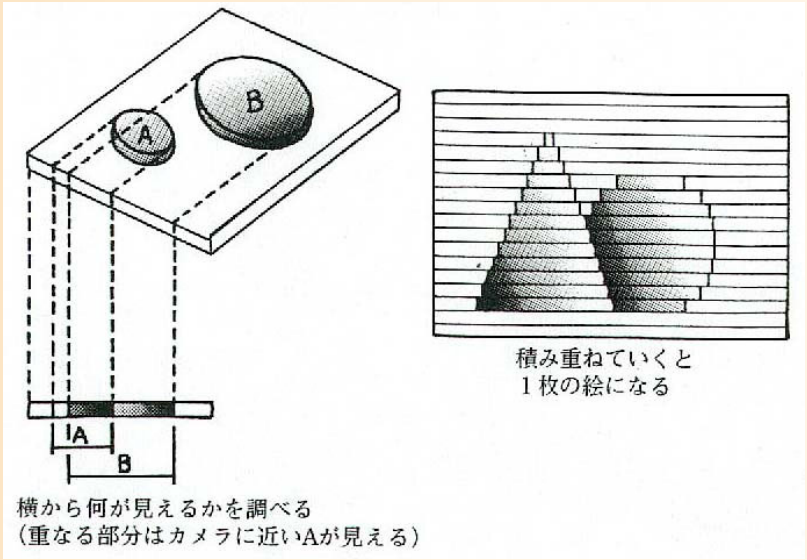


Zバッファの値（手前にあるほど明るく描画）

スキャンライン法

- 画像のそれぞれのラインごとに描画
- ラインの各スパンごとにどの線分が描画されるかを判定し、それぞれのスパンを描画

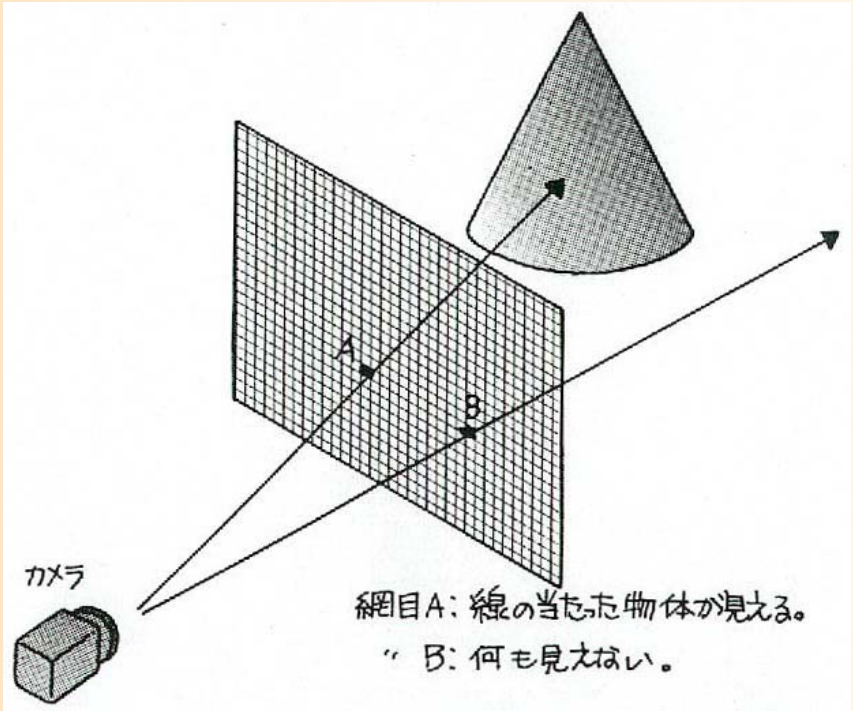




教科書 基礎知識 図2.31

レイトレーシング法

- 最も高品質な(最も実写に近い)画像が得られる方法
  - カメラからそれぞれのピクセルごとに視線方向に半直線(レイ)を飛ばし、物体との交差判定によりピクセルの色を計算



教科書 基礎知識 図2-33

レンダリング手法のまとめ

- Zソート
  - 面単位で描画、面単位で隠面消去
- Zバッファ
  - 面単位で描画、ピクセル単位で隠面消去
- スキャンライン
  - ライン単位で描画、ピクセル単位※で隠面消去
- レイトレーシング
  - ピクセル単位で描画、ピクセル単位※で隠面消去

※ 実際には、さらに細かい単位で描画が可能

サンプリング

- サンプリング（Sampling）
  - コンピュータグラフィックスでは、ピクセル単位で処理を行う
    - 本来は連続的な映像を、ピクセルごとに離散化して扱うことになる
  - そのままではカメラや人間の眼とは異なる画像となってしまうことがある
    - エイリアシング、モーションブラー、被視界深度
    - このような問題を解決するために、サンプリングの方法を工夫する技術がある

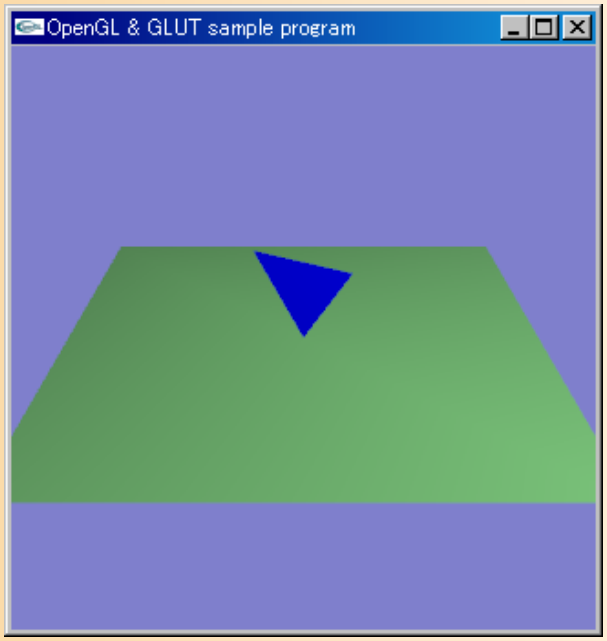
サンプリングの関連技術

- アンチエイリアシング
- モーションブラー
- 被視界深度

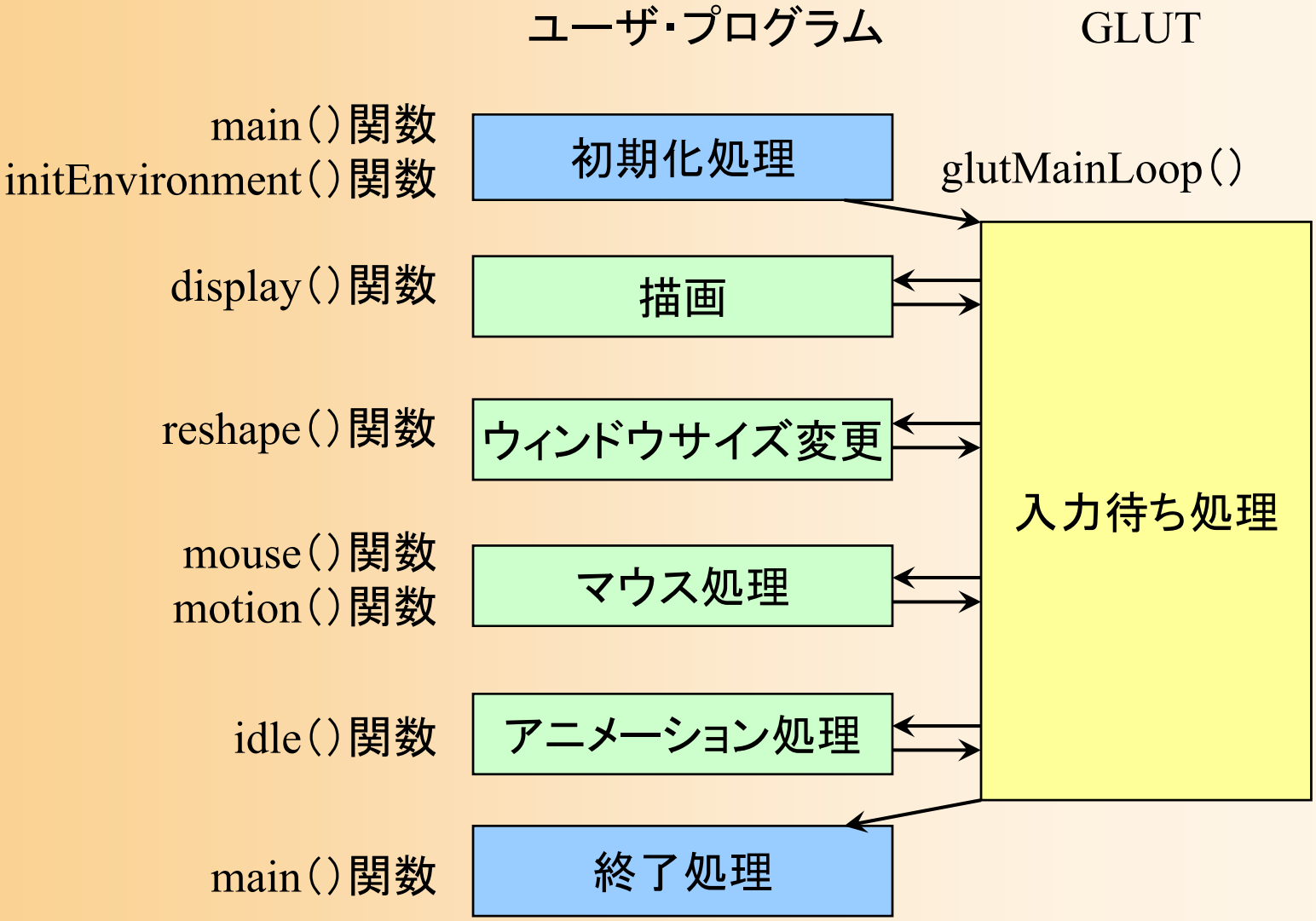
前回の演習の復習

サンプルプログラム

- opengl_sample.c
  - 地面と1枚の青い三角形が表示される
  - マウスの右ボタンドラッグで、視点を上下に回転



サンプルプログラムの構成



描画関数の流れ

```
//
//  ウィンドウ再描画時に呼ばれるコールバック関数
//
void  display( void )
{
        // 画面をクリア（ピクセルデータとZバッファの両方をクリア）
        // 変換行列を設定（ワールド座標系→カメラ座標系）
        // 光源位置を設定（モデルビュー行列の変更にあわせて再設定）
        // 地面を描画
        // 変換行列を設定（物体のモデル座標系→カメラ座標系）
        // 物体（１枚のポリゴン）を描画
        // バックバッファに描画した画面をフロントバッファに表示
}
```

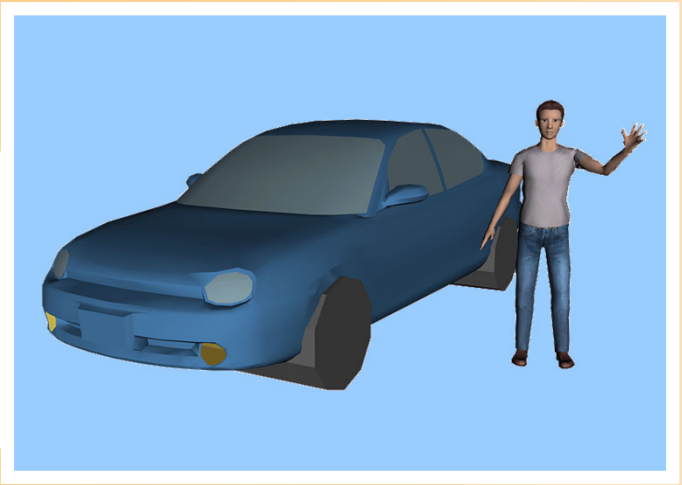
レンダリング・パイプライン

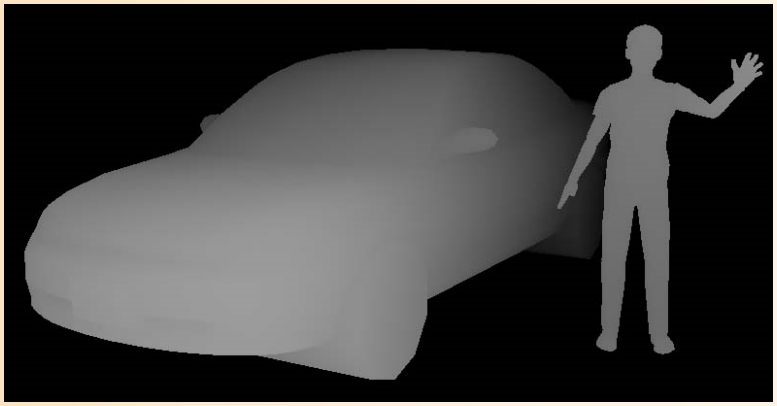
レンダリングの仕組み

- ポリゴンモデルによるモデリング（形状表現）
- Zバッファ法によるレンダリング（描画）
  - 現在、パソコンなどで最も広く使われている手法
  - OpenGL, DirectX などもZバッファ法を使用
  - 実用に使う可能性が高い
- 今回の講義の内容
  - Zバッファ法を使ったポリゴンモデルのレンダリングについて、もう少し詳しい仕組みを説明する
  - 今回の内容を踏まえて、次の演習を行う

Zバッファ法（復習）

- 画像とは別に、それぞれのピクセルの奥行き情報であるZバッファを持つ
- Zバッファを使うことで隠面消去を実現
  - すでに書かれているピクセルのZ座標と比較して、手前にある時のみピクセルを描画

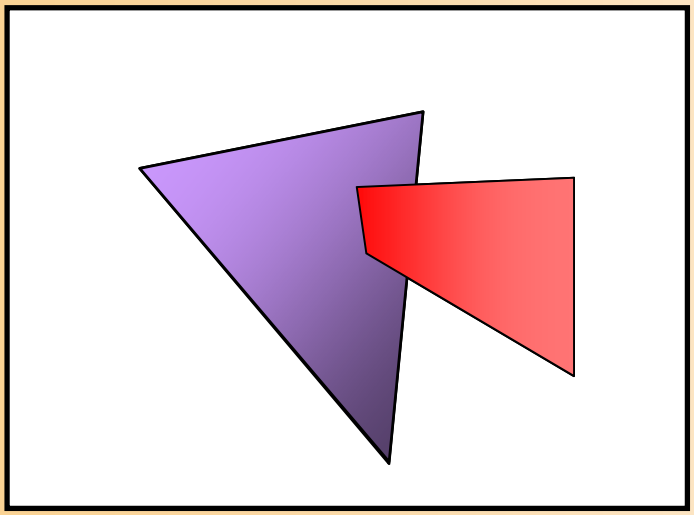


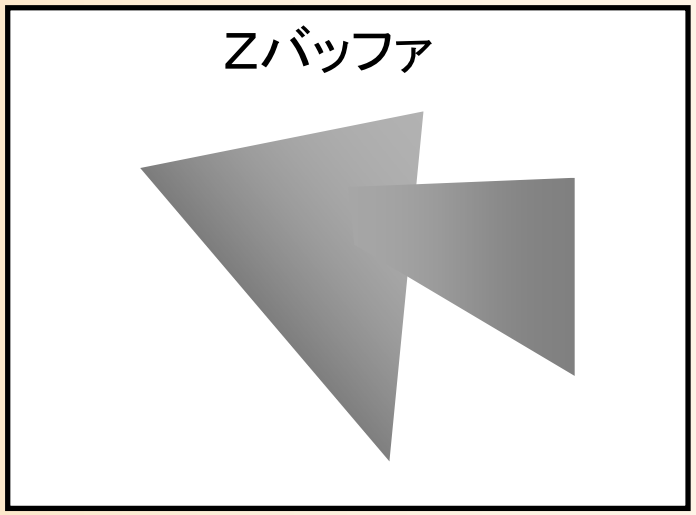


Zバッファの値（手前にあるほど明るく描画）

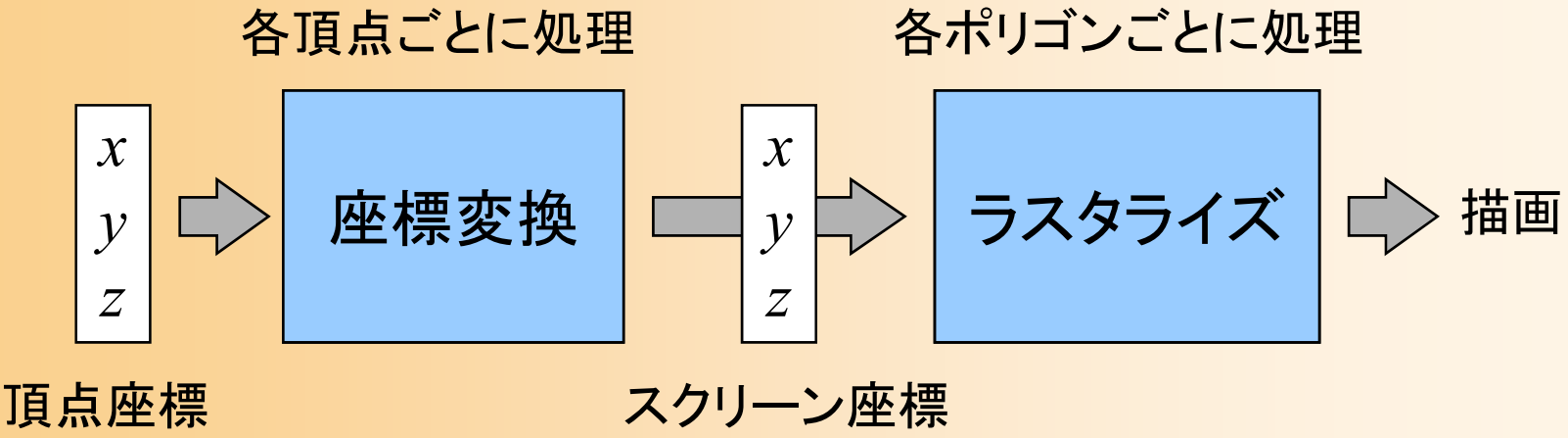
Zバッファ法による隠面消去（復習）

- Zバッファ法による面の描画
  - 面を描画するとき、各ピクセルの奥行き値（カメラからの距離）を計算して、Zバッファに描画
  - 同じ場所に別の面を描画するときは、すでに描画されている面より手前のピクセルのみを描画





レンダリング・パイプライン



- レンダリング・パイプライン（ビューイング・パイプライン、グラフィックス・パイプライン）
  - 入力されたデータを、流れ作業（パイプライン）で処理し、順次、画面に描画
  - ポリゴンのデータ（頂点データの配列）を入力
  - いくつかの処理を経て、画面上に描画される

レンダリング・パイプラインの利用

- OpenGL や DirectX などのライブラリを使用する場合は、この処理はライブラリが担当
  - レンダリング・パイプラインの処理を、自分でプログラミングする必要はない
- 自分のプログラムからは、適切な設定と、入力データの受け渡しを行なう
  - レンダリング・パイプラインの処理をきちんと理解していなければ、使いこなせない
  - ライブラリの使い方も理解する必要がある

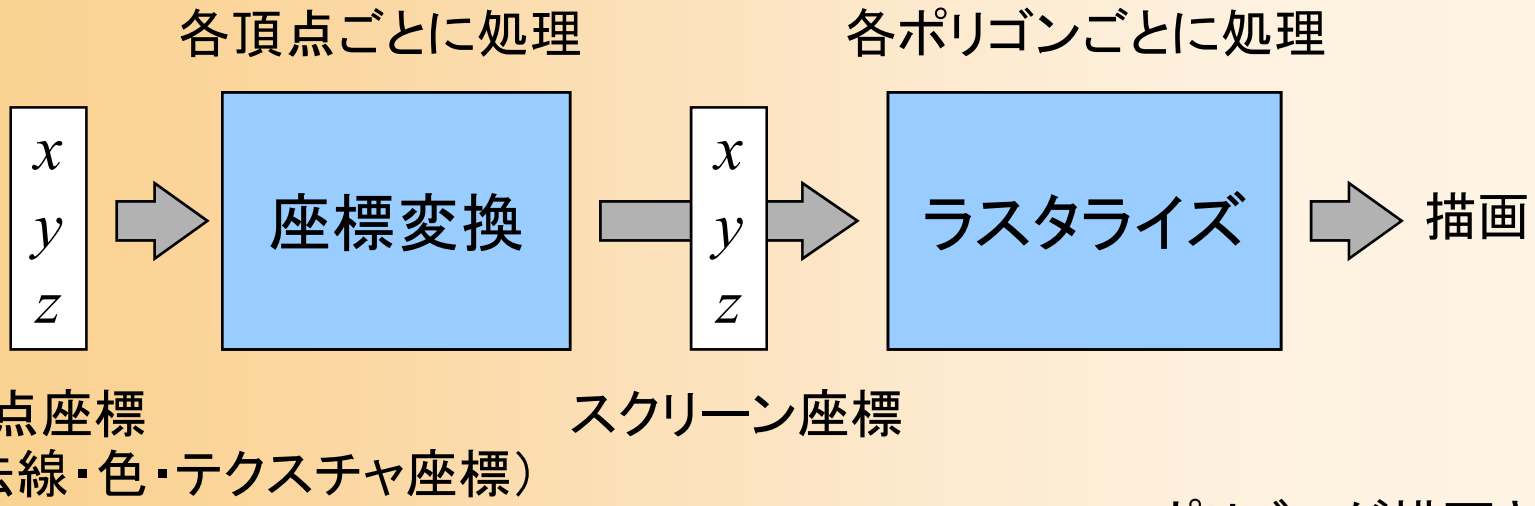
GLUTでのZバッファ法の利用

- 最初のウィンドウ生成時に、Zバッファを使用するように設定

```
int  main( int argc, char ** argv )
{
        // GLUTの初期化
        glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGBA  |
        ……                                          GLUT_DEPTH);
}
```

- Zバッファを有効にした状態で、OpenGLの関数を使用してポリゴンの描画を行うと、自動的にZバッファ法を使用しながら描画される

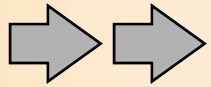
入出力の例（サンプルプログラム）



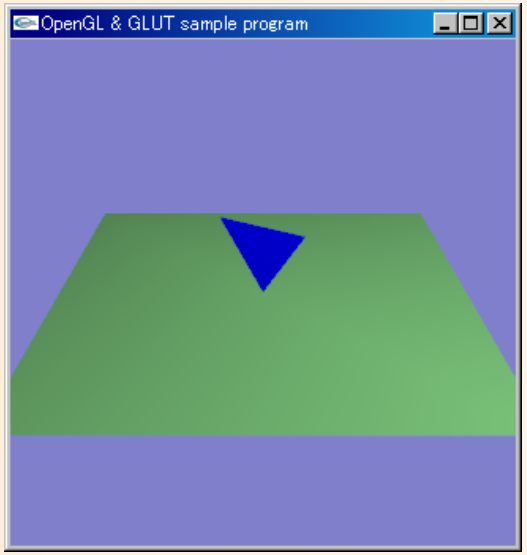
OpenGLにポリゴンの頂点情報を入力

```
glBegin( GL_TRIANGLES );
        glColor3f( 0.0, 0.0, 1.0 );
        glNormal3f( 0.0, 0.0, 1.0 );
        glVertex3f(-1.0, 1.0, 0.0 );
        glVertex3f( 0.0,-1.0, 0.0 );
        glVertex3f( 1.0, 0.5, 0.0 );
glEnd();
```

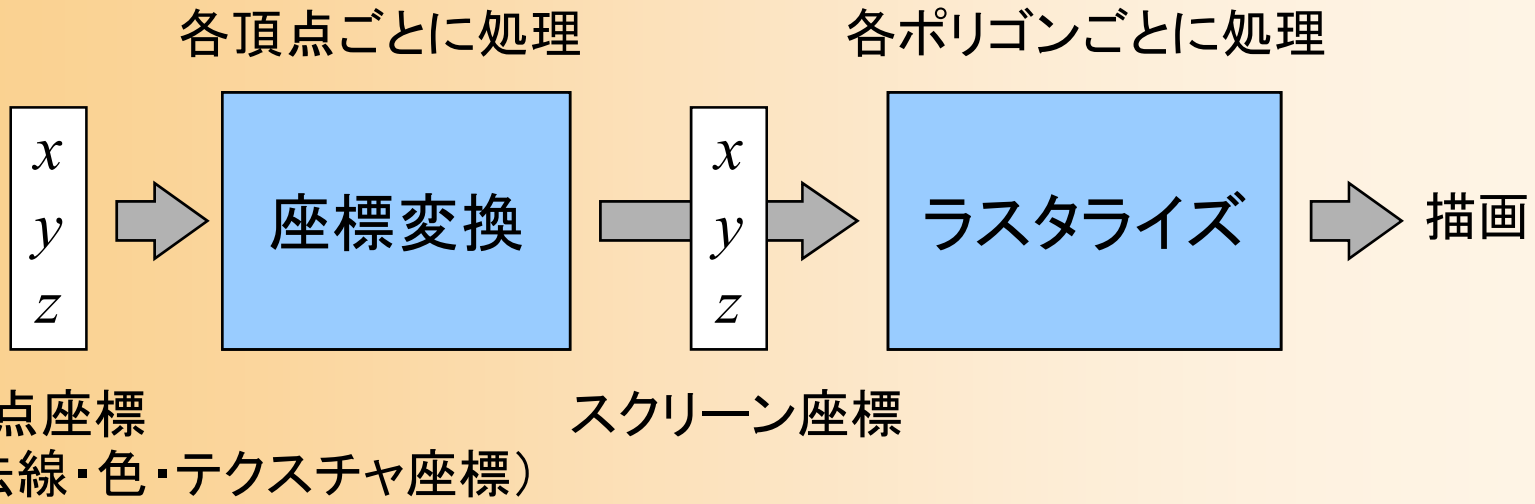
座標変換 &
ラスタライズ



ポリゴンが描画される

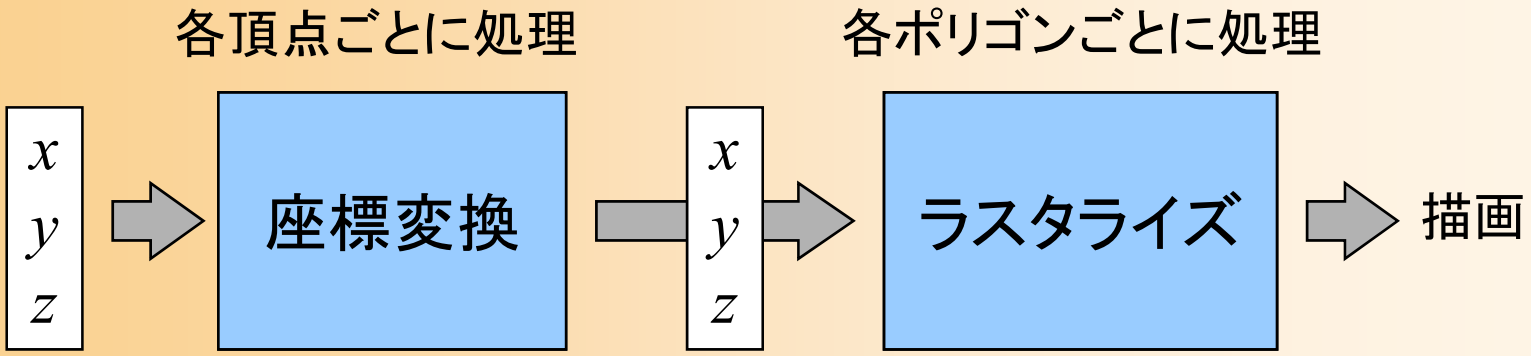


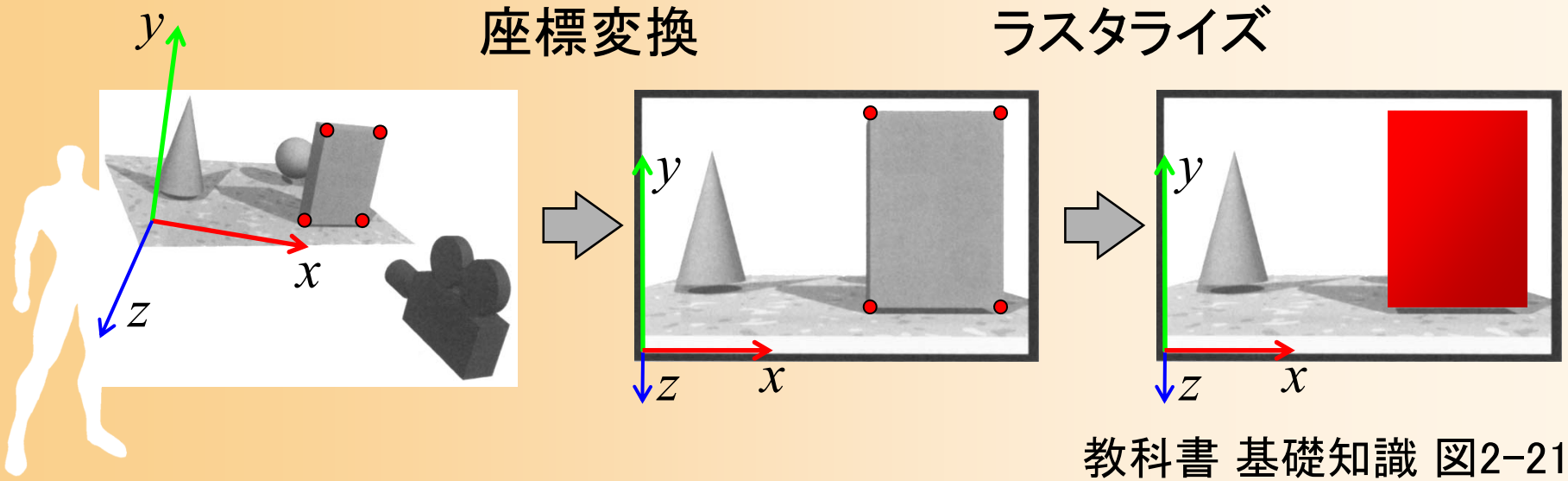
処理の流れ



- レンダリング時のデータ処理の流れ
  1. ポリゴンを構成する頂点の座標、法線、色、テクスチャ座標などを入力
  2. スクリーン座標に変換（座標変換）
  3. ポリゴンをスクリーン上に描画（ラスタライズ）

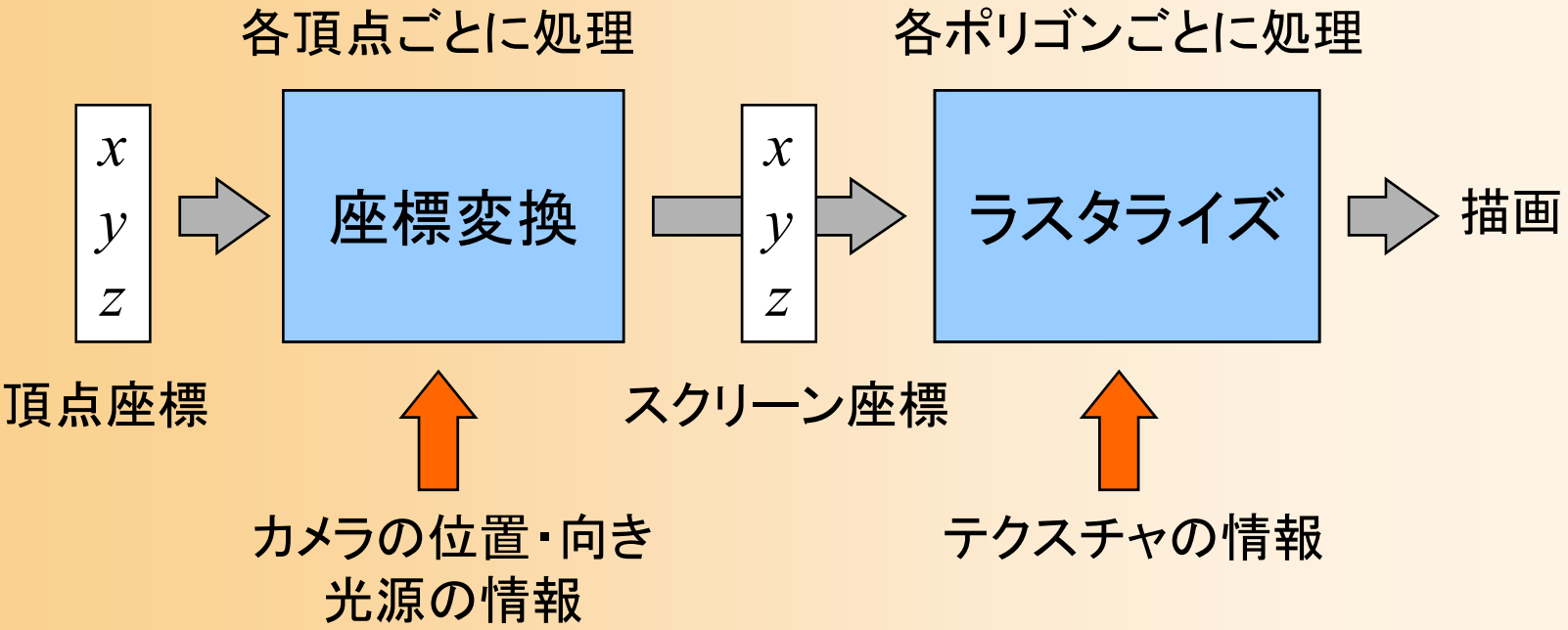
処理の流れ





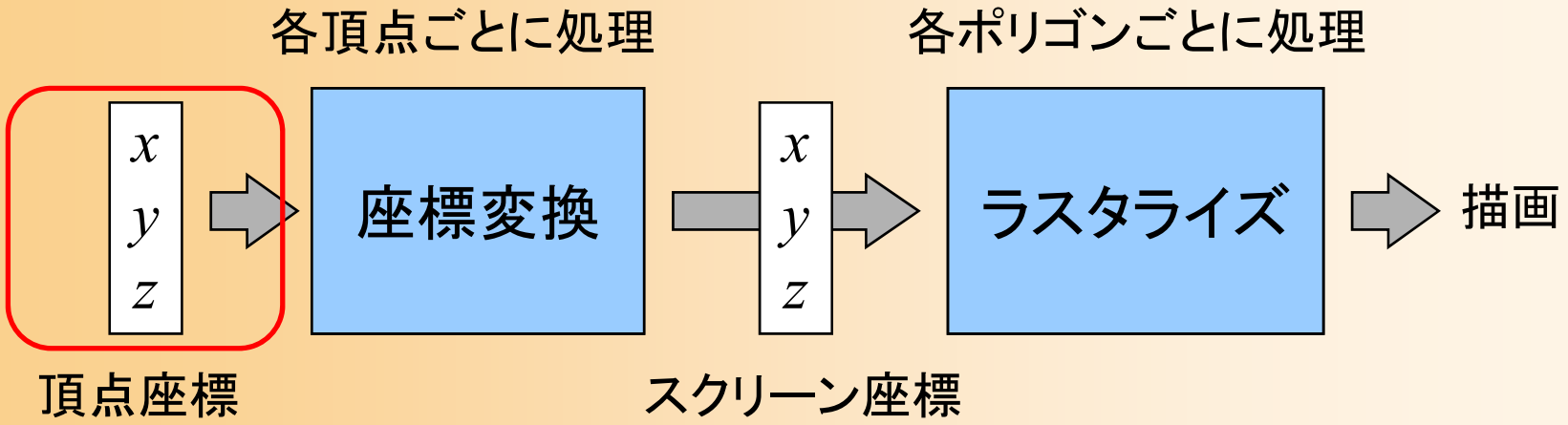
教科書 基礎知識 図2-21

描画前に行なう設定



- カメラの位置・向き（変換行列）の設定
- 光源の情報（位置・向き・色など）を設定
- テクスチャの情報を設定
- これらの情報は、次に更新されるまで記録される

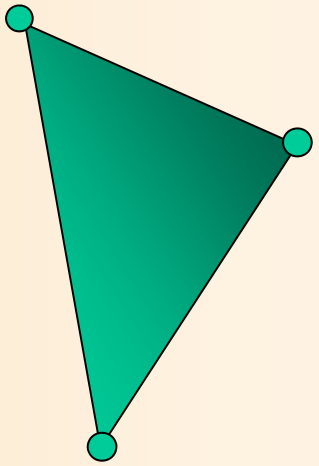
描画データの入力



- 物体の情報を入力
  - ポリゴンを構成する頂点の座標・法線・色・テクスチャ座標などを入力
- 表面の素材などを途中で変える場合は、適宜設定を変更

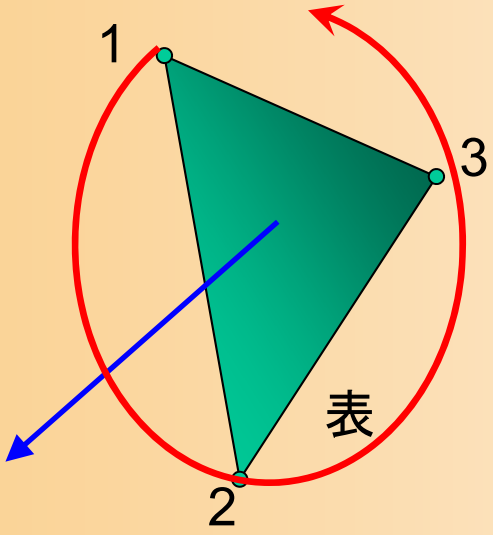
ポリゴンデータ

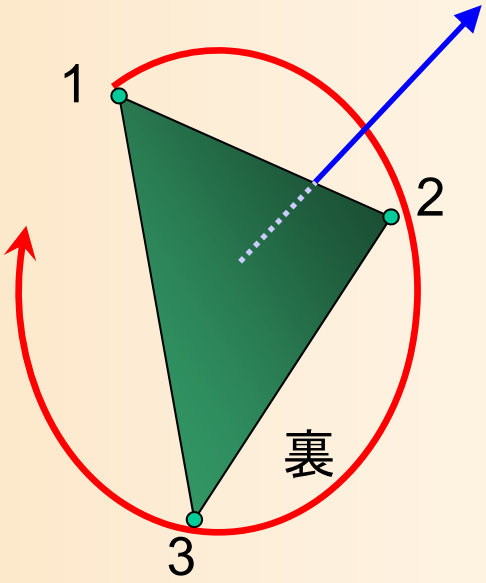
- ポリゴンの持つ情報
  - 各頂点の情報
    - 座標
    - 法線
    - 色
    - テクスチャ座標
      - 法線・テクスチャ座標については、詳細は後日の講義で説明
  - 面の向き
    - 頂点の順番によって面の向きを表す
    - 反時計回りに見える方が表（設定で向きは変更可）



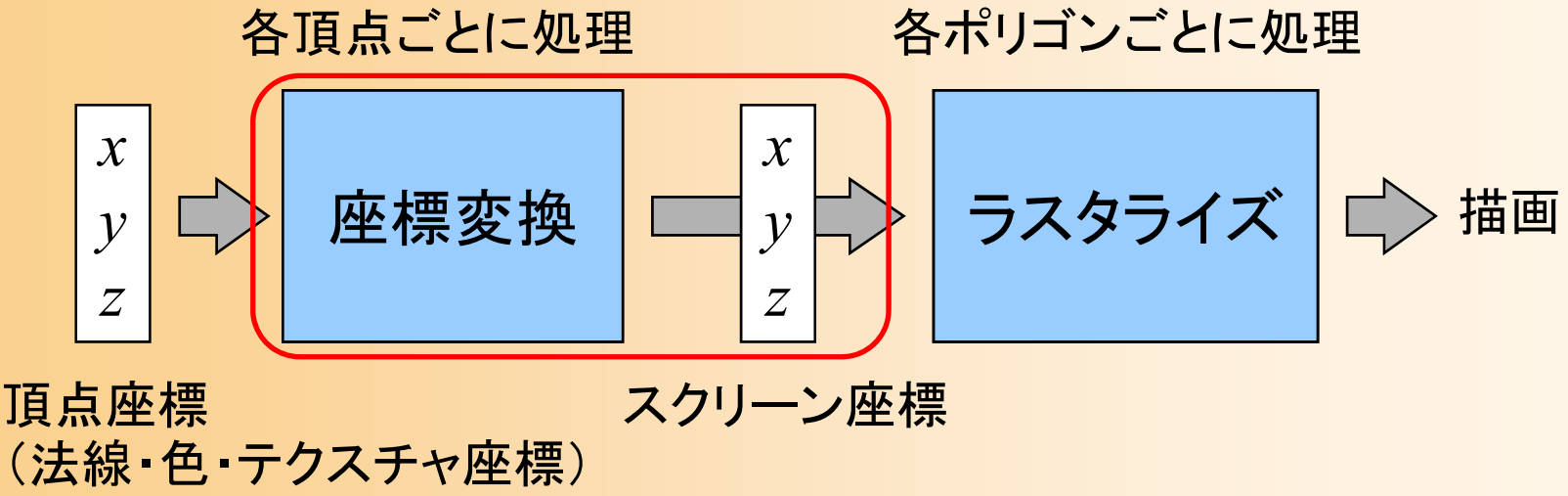
ポリゴンの向き（復習）

- 頂点の順番により、ポリゴンの向きを決定
  - 表から見て反時計回りの順序で頂点を与える
  - 視点と反対の向きでなら描画しない（背面除去）
    - 頂点の順序を間違えると、描画されないので、注意



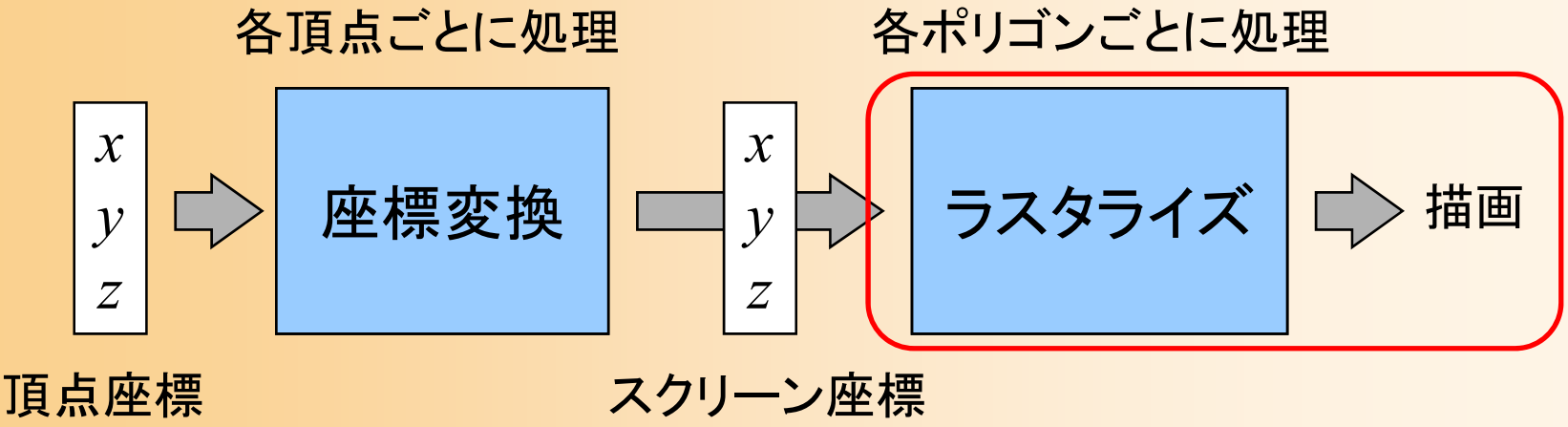


座標変換&ライト計算



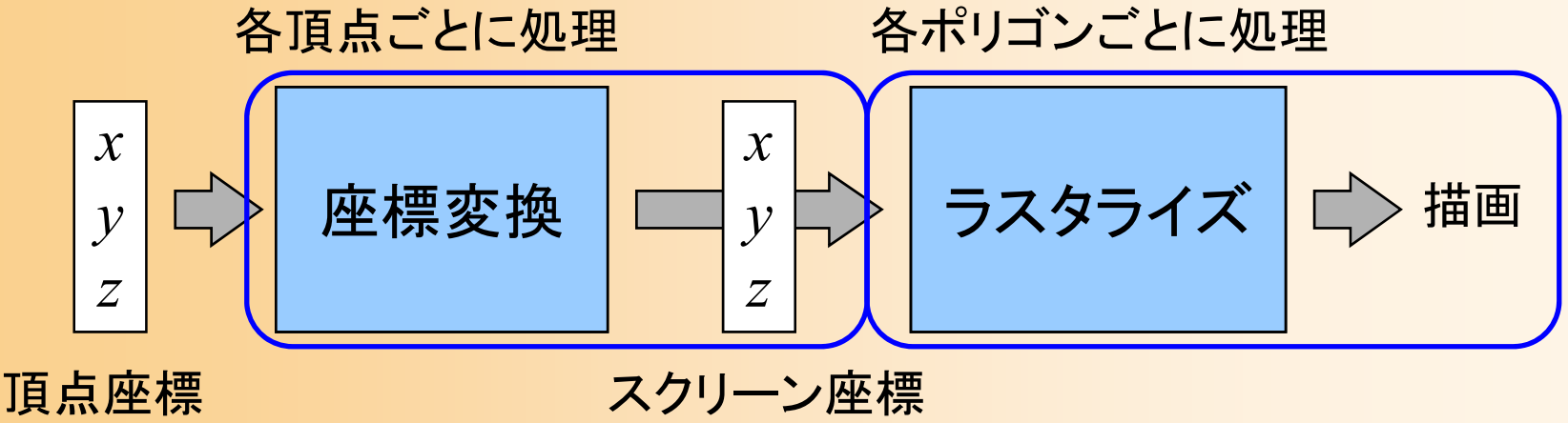
- 座標変換
  - 各頂点のスクリーン座標を計算
  - 法線と光源情報から、頂点の色を計算
    - 色の計算の方法については、後日の講義で説明
  - 面の向きをもとに背面除去、視界外の面も除去

ラスタライズ



- ラスタライズ（ポリゴンを画面上に描画）
  - グローシェーディングを適用
  - テクスチャマッピングを適用
  - Zバッファを考慮

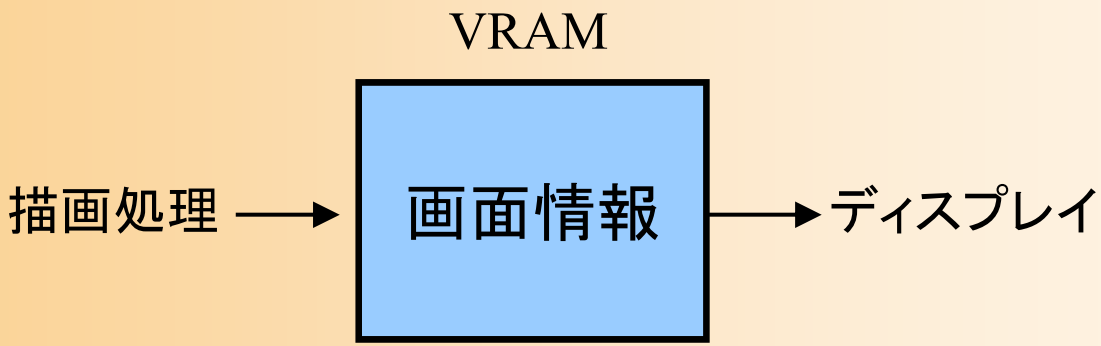
ハードウェアサポート



- ハードウェアによる処理
  - 昔はラスタライズのみをサポート
    - 使用可能なテクスチャの種類・枚数などは増えている
  - 現在は、座標変換や光の計算もハードウェアサポートされている
  - 最近では、ハードウェア処理の方法を変更できるようになっている（VertexShader, PixelShader）

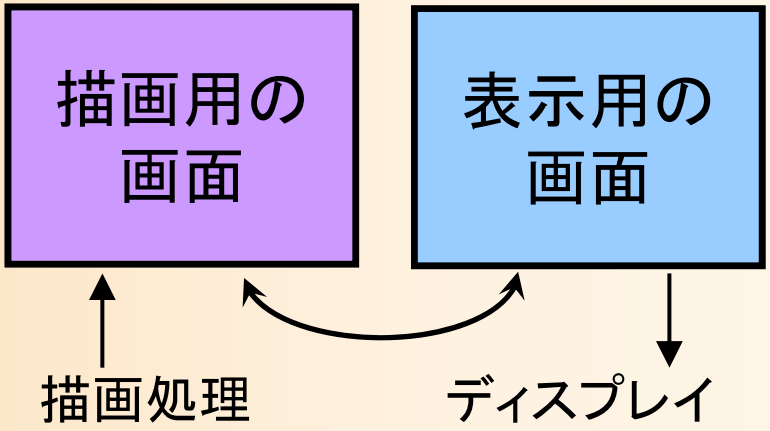
ダブルバッファリング

- 画面表示の仕組み
  - ビデオメモリ（VRAM）上の画面データをディスプレイ上に表示
  - 描画途中の画面を表示するとちらついてしまう
    - 描画量が少ない場合は垂直同期（VSYNC）中に描画すればちらつかない



ダブルバッファリング

- 2枚の画面を使用
  - 表示用
  - 描画用（＋Zバッファ）



- ダブルバッファリング
  - 描画用の画面に対して描画
  - 描画が完了したら、描画用の画面と表示用の画面を切り替える（もしくは、描画用の画面を表示用の画面にコピーする）

GLUTでのダブルバッファリング

- 最初にダブルバッファリングの使用を設定

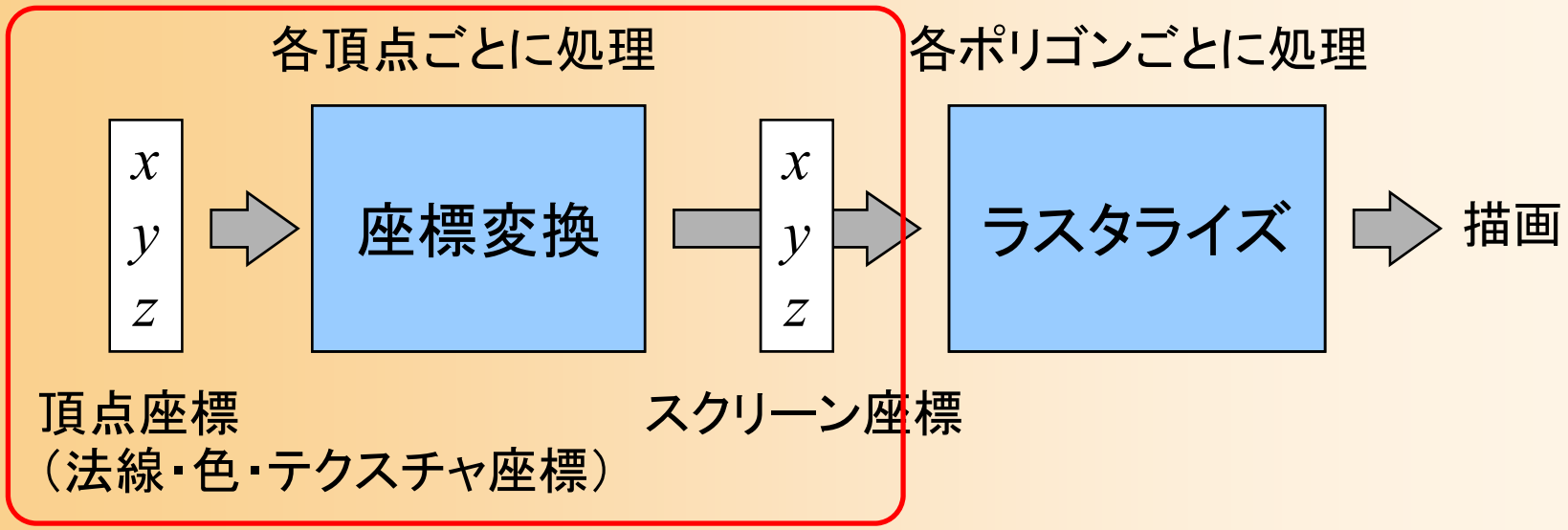
```
int  main( int argc, char ** argv )
{
        // GLUTの初期化
        glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGBA  |
        ……                                          GLUT_DEPTH);
}
```

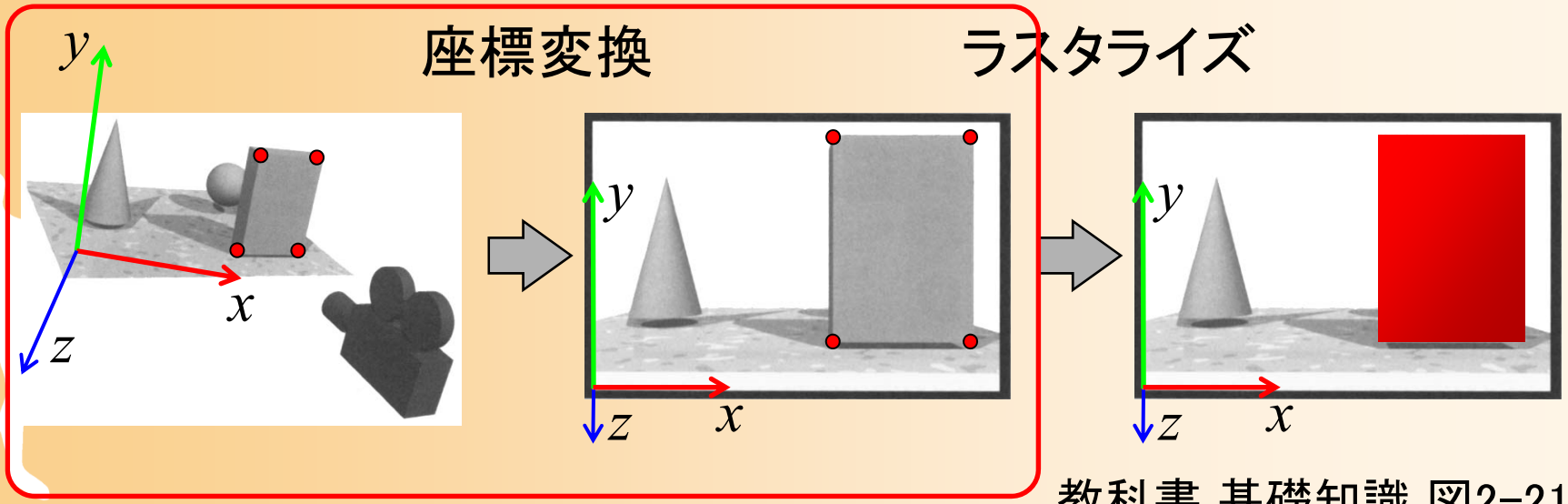
- 毎回の描画処理後に、画面の切り替えを実行

```
void  display( void )
{
        ……
        // バックバッファに描画した画面をフロントバッファに表示
        glutSwapBuffers();
}
```

座標変換

座標変換

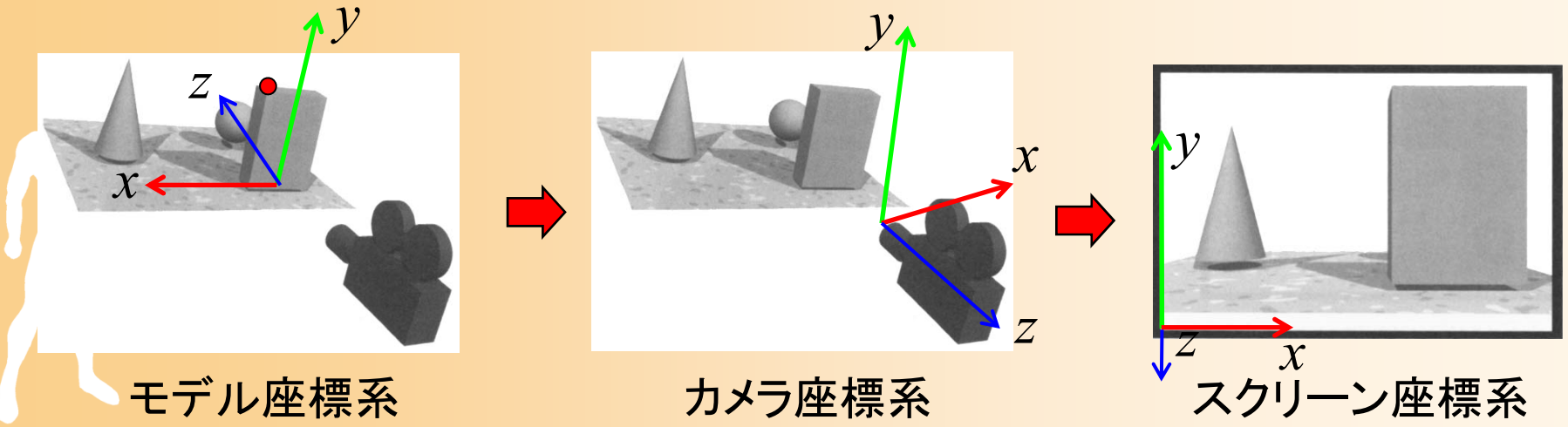




教科書 基礎知識 図2-21

座標変換

- 座標変換(Transformation)
  - 行列演算を用いて、ある座標系から、別の座標系に、頂点座標やベクトルを変換する技術
    - カメラから見た画面を描画するためには、モデルの頂点座標をカメラ座標系(最終的にはスクリーン座標系)に変換する必要がある



モデル座標系　　カメラ座標系　　スクリーン座標系

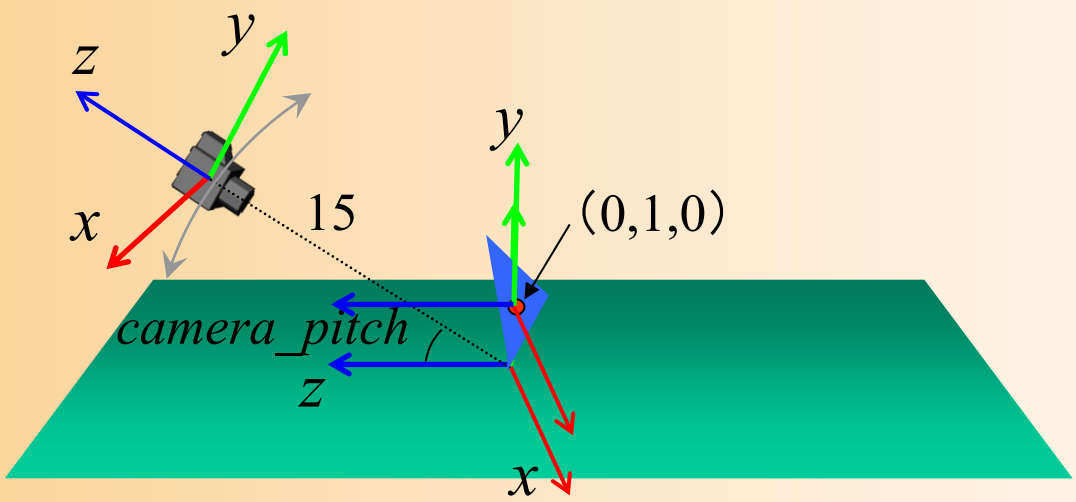
同次座標変換

- 同次座標変換
  - ４×４行列の演算により、３次元空間における平行移動・回転・拡大縮小（アフィン変換）などの操作を統一的に実現
    - (x, y, z, w)の４次元座標値（同次座標）を扱う
    - ３次元座標値は$(x/w, y/w, z/w)$で計算（通常は $w = 1$）

$$\begin{pmatrix} R_{00}S_x & R_{01} & R_{02} & T_x \\ R_{10} & R_{11}S_y & R_{12} & T_y \\ R_{20} & R_{21} & R_{22}S_z & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix}$$

変換行列の例

- 変換行列の詳しい使い方の説明は、後日



ポリゴンを基準とする座標系での頂点座標

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -15 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-camera\_pitch) & -\sin(-camera\_pitch) & 0 \\ 0 & \sin(-camera\_pitch) & \cos(-camera\_pitch) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}$$

カメラから見た頂点座標（描画に使う頂点座標）

プログラムの例

- 適切な変換行列を設定することで、カメラや物体の位置・向きを自在に変更できる

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -15 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-camera\_pitch) & -\sin(-camera\_pitch) & 0 \\ 0 & \sin(-camera\_pitch) & \cos(-camera\_pitch) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}$$

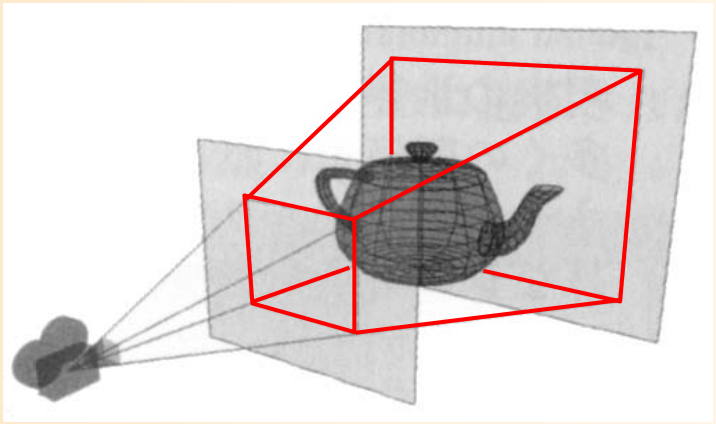
```
// 変換行列を設定（ワールド座標系→カメラ座標系）
glMatrixMode( GL_MODELVIEW );
glLoadIdentity();
glTranslatef( 0.0,  0.0, - 15.0 );
glRotatef( - camera_pitch, 1.0, 0.0, 0.0 );
glTranslatef( 0.0, 1.0, 0.0 );
```

座標変換の詳細

- 後日の講義で説明
- 非常に重要な原理

クリッピング＋背面除去

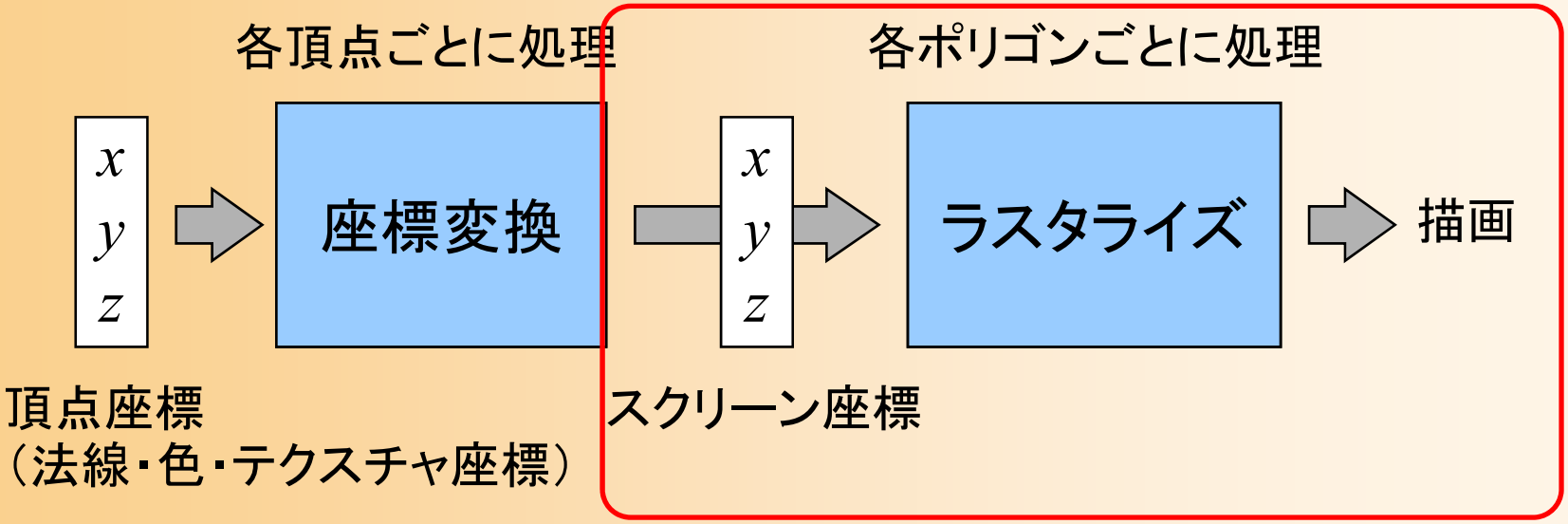
- クリッピング
  - 視野（ビューポート）外のポリゴンは描画しない

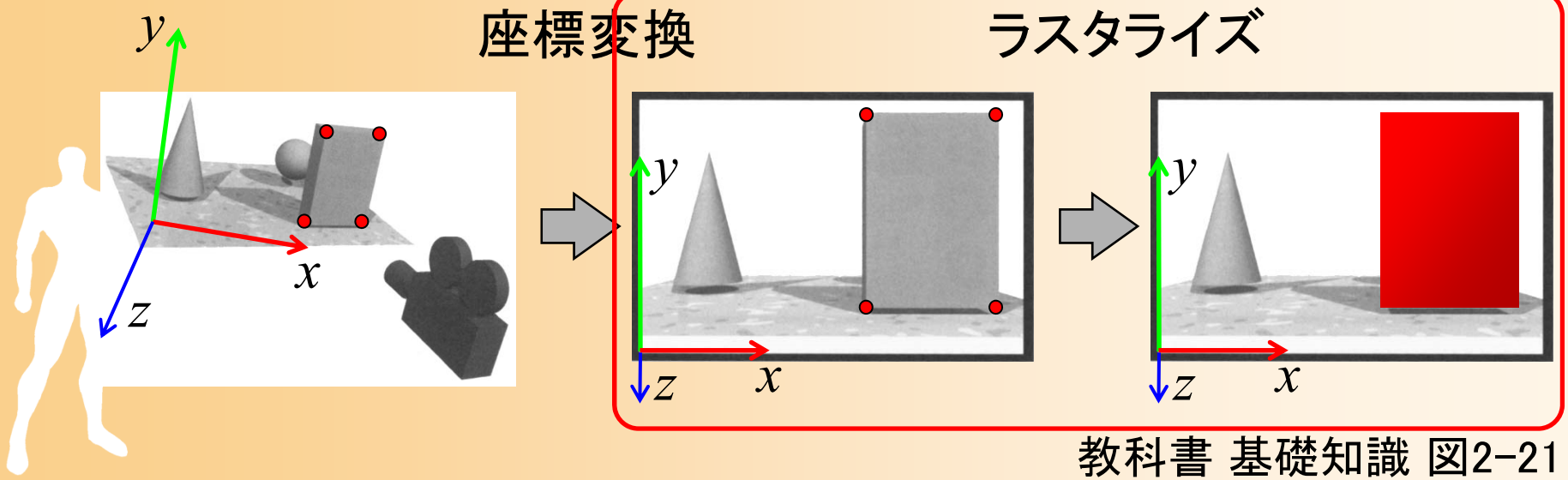


- 背面除去
  - カメラから見て後ろ向きのポリゴンは描画しない
- 座標変換後のポリゴンに対して、クリッピングと背面除去の判定を行う

ラスタライズ

ラスタライズ





教科書 基礎知識 図2-21

ラスタライズ

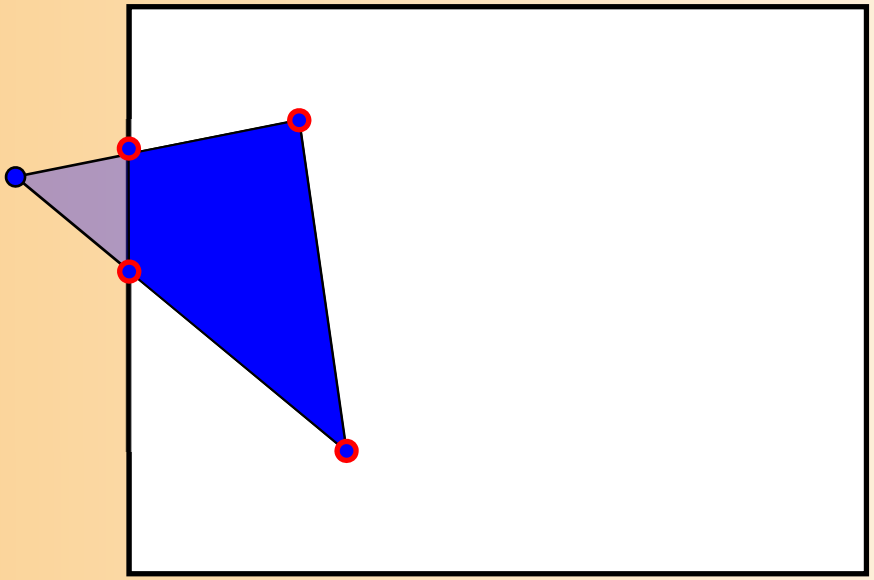
- ラスタライズ
  - スクリーン上にポリゴンを描画する処理
- ラスタライズの入力
  - ポリゴンの各頂点のスクリーン座標（x, y, z）・色（r, g, b）
  - テクスチャマッピング時は、さらに、各頂点のテクスチャ座標（t, v）

ラスタライズ

- クリッピング
- 三角面の描画
  - グローシェーディング、テクスチャマッピング、Zテスト、などを描画時に実行
    - グローシェーディング、テクスチャマッピングについては、後日の講義で説明

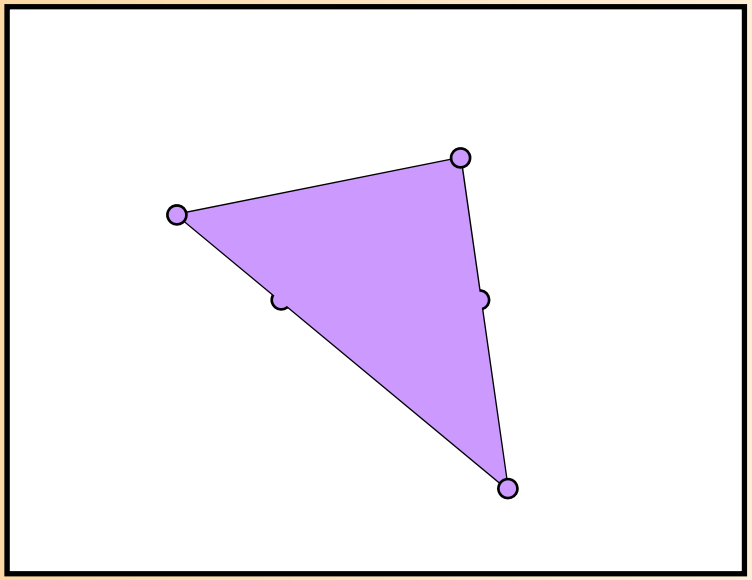
クリッピング

- クリッピング
  - ポリゴンの頂点が画面の外側に出ている時は、画面内部の部分だけが残るように分割



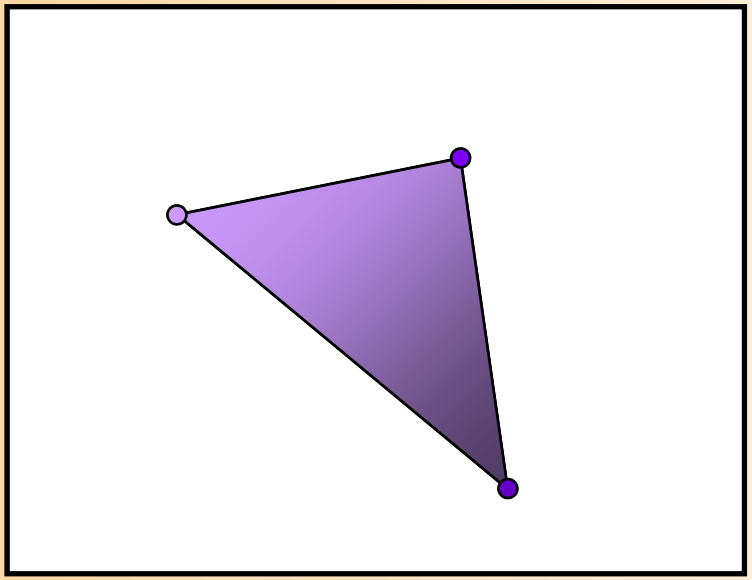
三角形の描画

- 各ラインごとに描画
  - 各ラインごとに両端のピクセルを計算
  - 両端点の間にあるピクセルを描画



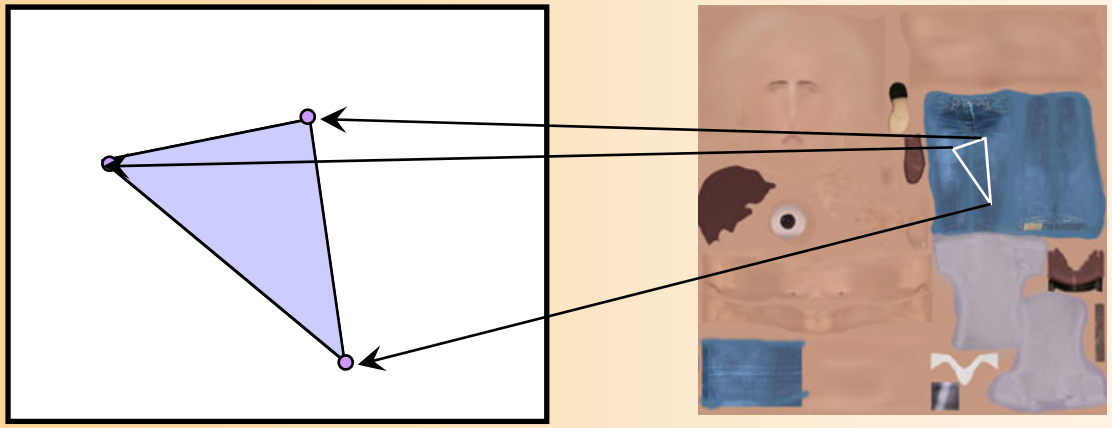
（グローシェーディング）

- 光の効果の計算の時に頂点の色を決定
- 頂点の色を補間して各ピクセルの色を計算



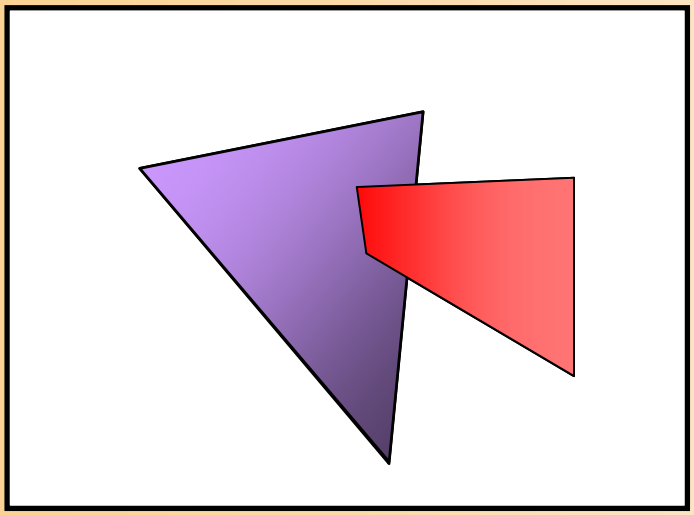
（テクスチャマッピング）

- 頂点ごとのテクスチャ座標が与えられる
- テクスチャ座標を補間してピクセルごとのテクスチャ座標を計算
  - 奥行きの影響が加わるように考慮
  - ピクセルがなめらかになるように補間

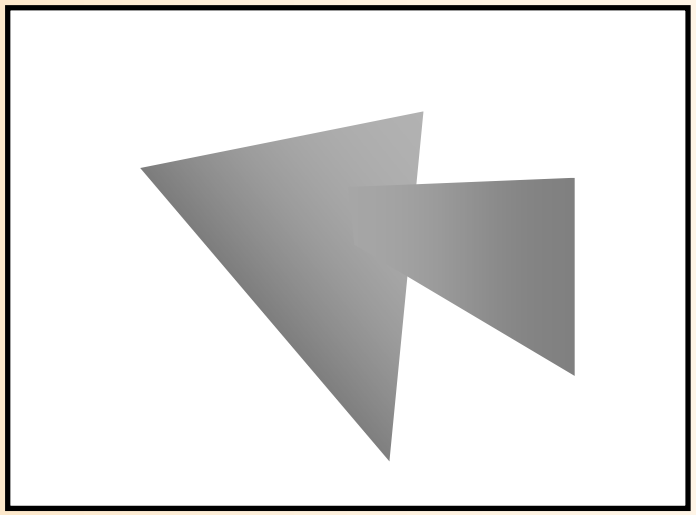


Zテスト

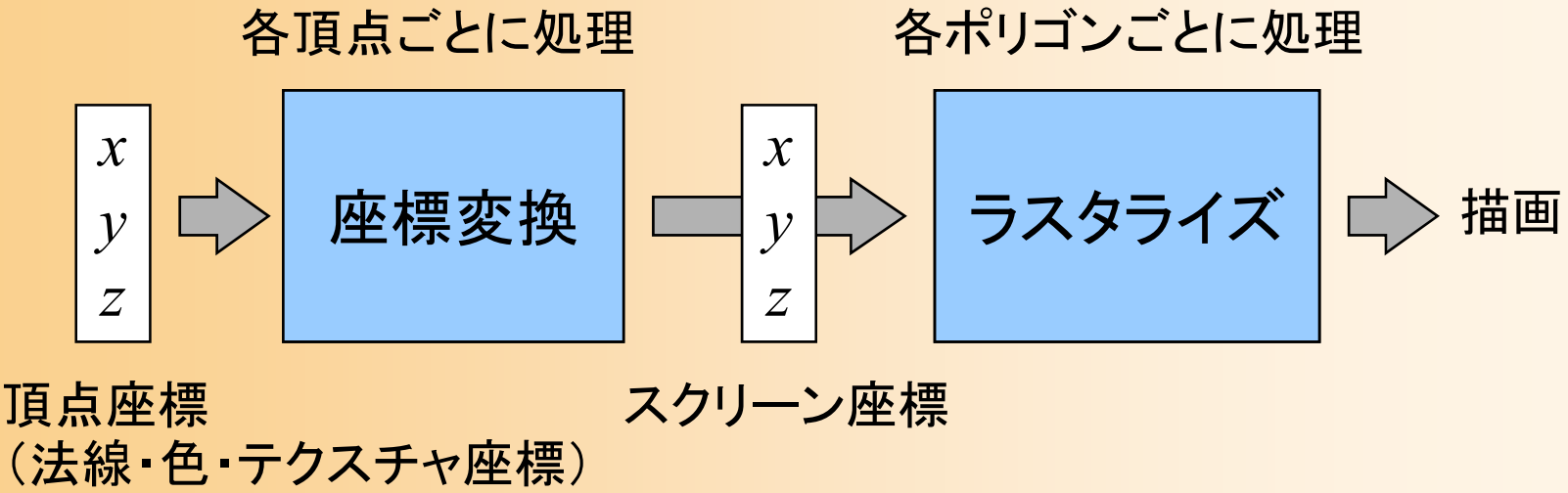
- Zバッファ法によるレンダリング
  - それぞれの頂点はZ座標値を持つ
  - 頂点のZ値を補間してピクセルごとのZ値を計算
  - 各ピクセルを描画時にZバッファのZ座標と比較



Zバッファ



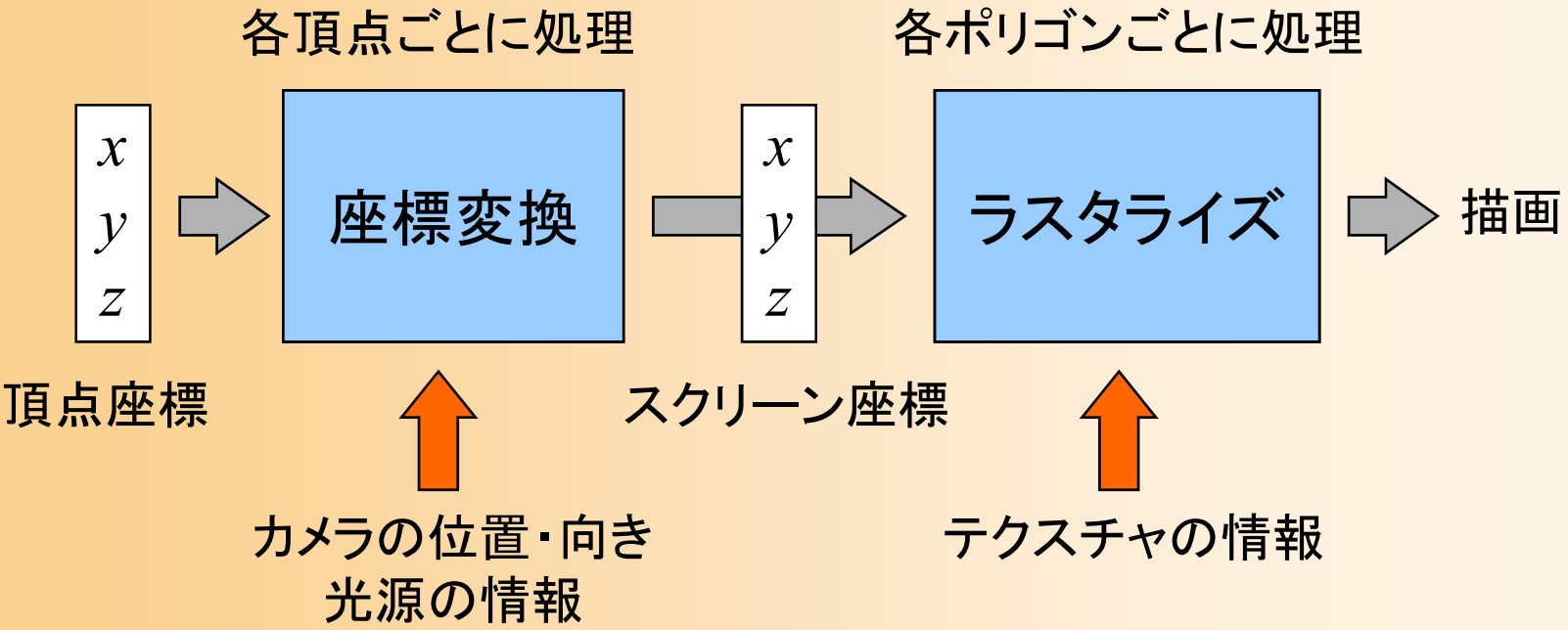
レンダリング・パイプラインのまとめ



- レンダリング時のデータ処理の流れ
  1. ポリゴンを構成する頂点の座標、法線、色、テクスチャ座標などを入力
  2. スクリーン座標に変換（座標変換）
  3. ポリゴンをスクリーン上に描画（ラスタライズ）

OpenGLでの
レンダリングの設定

レンダリング・パイプラインの設定（復習）



- 描画の前に、さまざまな設定を行うことができる
- 各機能を使うかどうか（Zバッファ、背面除去等）
- カメラの位置・向き（変換行列）の設定
- 光源の情報（位置・向き・色など）を設定

描画機能の設定

- さまざまな描画機能のオン･オフを設定
  - 不必要な処理はオフにすることで、高速できる
  - 初期状態ではオフになっている機能が多いので、必要な機能はオンに設定する必要がある
- glEnable（機能の種類）, glDisable（･･･）
  - 各機能のオン･オフを変更する
    - GL_LIGHTING, GL_COLOR_MATERIAL, GL_DEPTH_TEST, CL_CULL_FACE, etc
  - 各機能の動作はそれぞれ別の関数で設定

サンプルプログラムの描画機能の設定

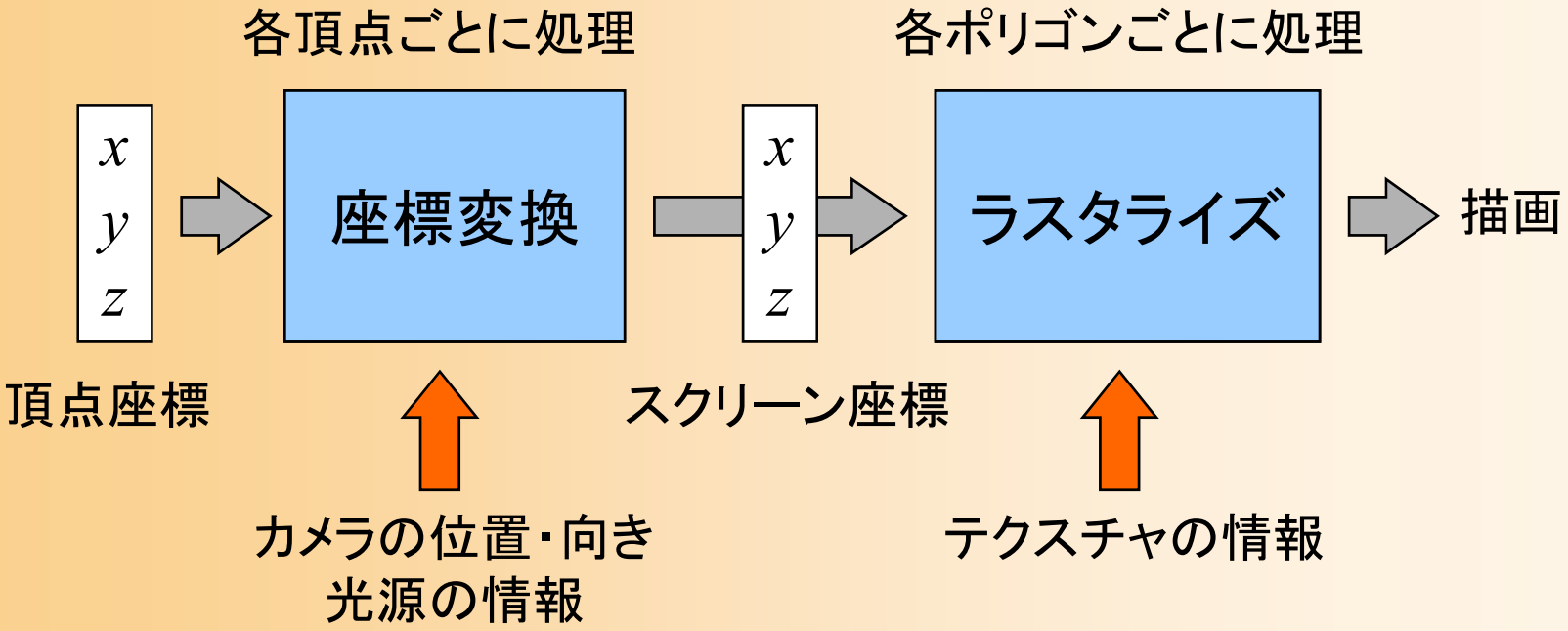
- 標準的な描画の設定（最初に一度だけ設定）

```
void  initEnvironment( void )
{
        ......
        // 光源計算を有効にする
        glEnable( GL_LIGHTING );

        // 物体の色情報を有効にする
        glEnable( GL_COLOR_MATERIAL );

        // Zテストを有効にする
        glEnable( GL_DEPTH_TEST );

        // 背面除去を有効にする
        glCullFace( GL_BACK );
        glEnable( GL_CULL_FACE );

        // 背景色を設定
        glClearColor( 0.5, 0.5, 0.8, 0.0 );
}
```

描画機能の設定（その他）

- 背面除去の設定
  - glCullFace(GL_BACK)
  - 表面・背面のどちらを描画しないかを設定
- 背景色の設定
  - glClearColor(r, g, b, a)
  - 画面をクリアしたときの色を設定

その他の設定



- 光源の情報（位置・向き・色など）を設定
- テクスチャマッピングの設定
  - これらについては、後日の講義・演習で説明

レンダリング

- Zバッファ法によるレンダリング
  - 基本的には、OpenGLが自動的にZバッファ法を用いたレンダリングを行うので、自分のプログラムでは特別な処理は必要ない
  - 最初に、Zテストを有効にするように、設定する必要がある

```
void  initEnvironment( void )
{
        ......

        // Zテストを有効にする
        glEnable( GL_DEPTH_TEST );

        ......
}
```

OpenGLでの
ポリゴンの描画方法

ポリゴンの描画

- glBegin() ～ glEnd() 関数を使用

```
glBegin( 図形の種類 );
        この間に図形を構成する頂点データを指定
glEnd();
```

※ 頂点データの指定では、一つの関数で、図形を構成する頂点の座標・色・法線などの情報の一つを指定

- 図形の種類（各種の点・線・面が指定可能）
  - GL_POINTS（点）、GL_LINES（線分）、GL_TRIANGLES（三角面）、GL_QUADS（四角面）、GL_POLYGON（ポリゴン）、他

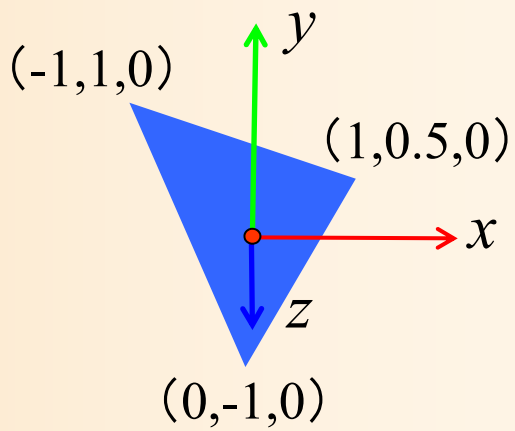
頂点データの指定

- glColor3f(r, g, b)
  - これ以降の頂点の色を設定
- glNormal3f(nx, ny, nz)
  - これ以降の頂点の法線を設定
- glVertex3f(x, y, z)
  - 頂点座標を指定
  - 色・法線は、最後に指定したものが使用される

ポリゴンの描画の例(1)

- 1枚の三角形を描画
  - 各頂点の頂点座標、法線、色を指定して描画
  - ポリゴンを基準とする座標系(モデル座標系)で頂点位置・法線を指定

```
glBegin( GL_TRIANGLES );
        glColor3f( 0.0, 0.0, 1.0 );
        glNormal3f( 0.0, 0.0, 1.0 );
        glVertex3f(-1.0, 1.0, 0.0 );
        glVertex3f( 0.0,-1.0, 0.0 );
        glVertex3f( 1.0, 0.5, 0.0 );
glEnd();
```

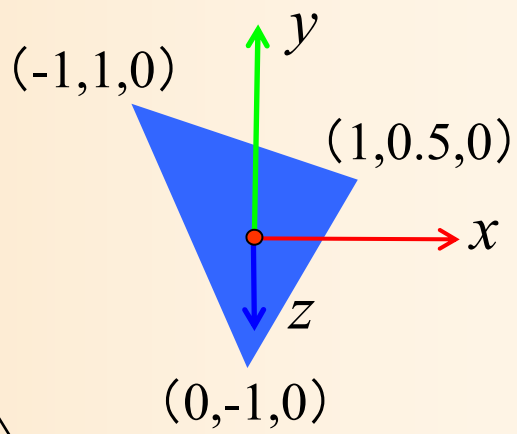


GL_TRIANGLES が指定されているので、3つの頂点をもとに、1枚の三角面を描画 (6つの頂点が指定されたら、2枚描画)

ポリゴンの描画の例(1)

- 頂点の色・法線は、頂点ごとに指定可能
  - 指定しなければ、最後に指定したものが使われる

```
glBegin( GL_TRIANGLES );
        glColor3f( 0.0, 0.0, 1.0 );
        glNormal3f( 0.0, 0.0, 1.0 );
        glVertex3f(-1.0, 1.0, 0.0 );

        glColor3f( 0.0, 0.0, 1.0 );
        glNormal3f( 0.0, 0.0, 1.0 );
        glVertex3f( 0.0,-1.0, 0.0 );

        glColor3f( 0.0, 0.0, 1.0 );
        glNormal3f( 0.0, 0.0, 1.0 );
        glVertex3f( 1.0, 0.5, 0.0 );
glEnd();
```

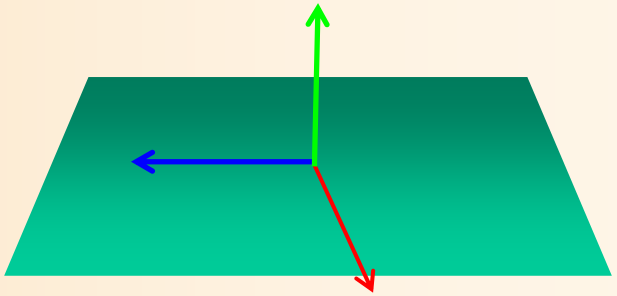


省略可能
(前の頂点と同じ色・法線であれば)

ポリゴンの描画の例(2)

- 1枚の四角形として地面を描画
  - 各頂点の頂点座標、法線、色を指定して描画
  - 真上(0,1,0)を向き、水平方向の長さ10の四角形

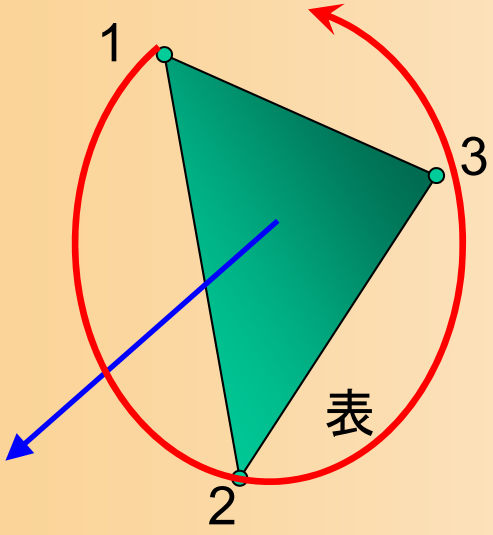
```
// 地面を描画
glBegin( GL_POLYGON );
        glNormal3f( 0.0, 1.0, 0.0 );
        glColor3f( 0.5, 0.8, 0.5 );

        glVertex3f( 5.0, 0.0, 5.0 );
        glVertex3f( 5.0, 0.0,-5.0 );
        glVertex3f(-5.0, 0.0,-5.0 );
        glVertex3f(-5.0, 0.0, 5.0 );
glEnd();
```

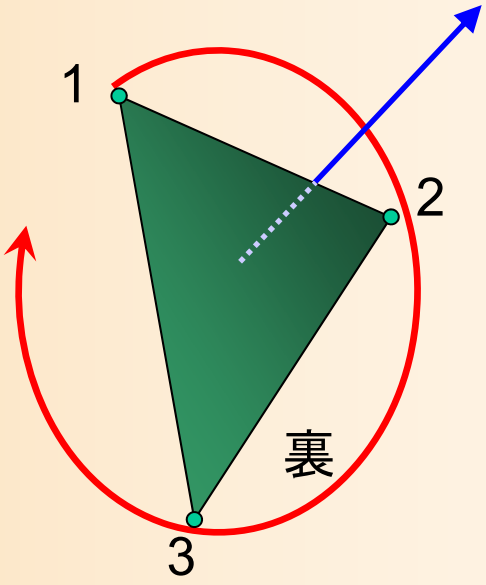


GL_POLYGON が指定されているので、n 個の頂点をもとに、n 角面を描画
(1度に1枚しか描画できない)

ポリゴンの向き

- 頂点の順番により、ポリゴンの向きを決定
  - 表から見て反時計回りの順序で頂点を与える
  - 視点と反対の向きでなら描画しない（背面除去）
    - 頂点の順序を間違えると、描画されないので、注意



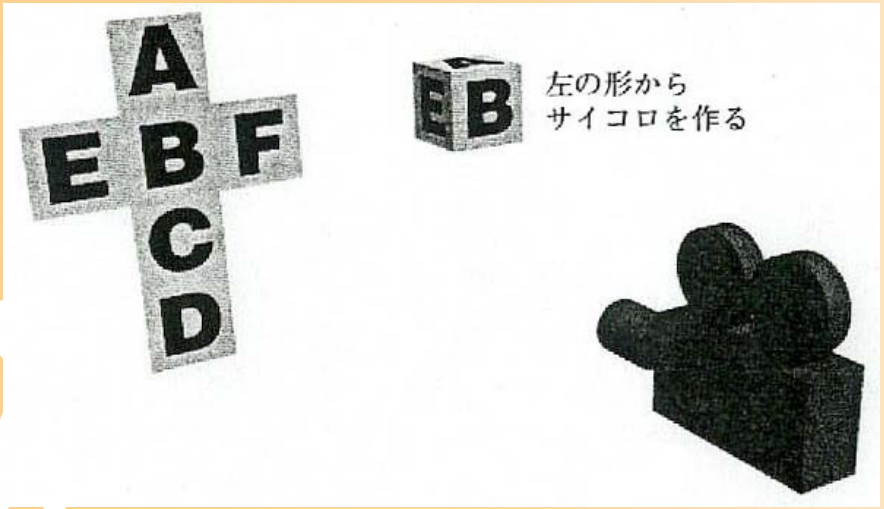


背面消去(復習)

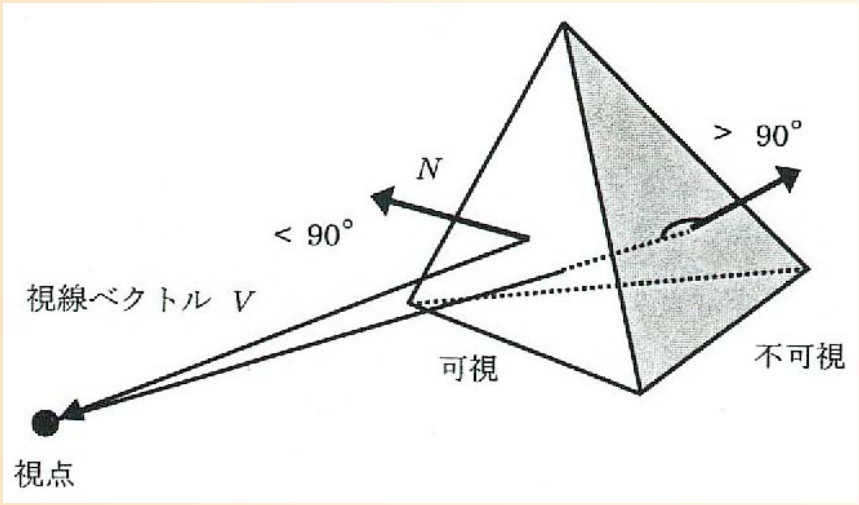
- 背面消去(後面消去、背面除去、後面除去)
  - バックフェースカリング、とも呼ぶ
- 後ろ向きの面の描画を省略する処理
- サーフェスモデルであれば、後ろ向きの面は描画は不要である点に注目する
  - 仮に描画したとしても、その後、手前側にある面で上書きされる
  - 裏向きの面の描画を省略することで処理を高速化できる(単純に考えると、約半分に減らせる)

背面消去(復習)

- 後ろ向きの面の判定方法
  - 視線ベクトル(カメラから面へのベクトル)と面の法線ベクトルの内積により判定

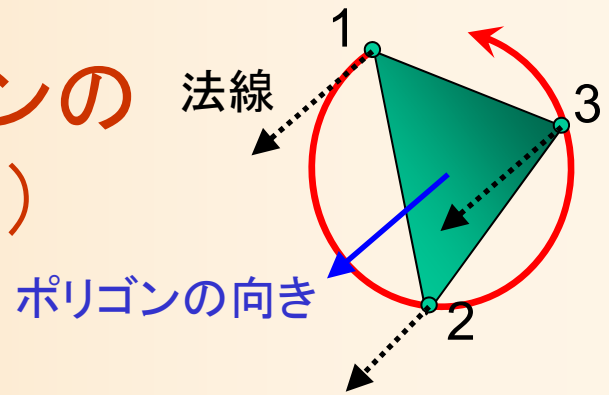


教科書 基礎知識 図2-22



教科書 基礎と応用 図3.5

法線とポリゴンの向き



- OpenGLでは、法線とポリゴンの向きは、独立の扱い（要注意）
- 法線
  - 頂点ごとに、関数（glNormal3f()）により指定
  - 光のモデルに従って色を計算するために使用（詳細は後日の講義で説明）
- ポリゴンの向き
  - ポリゴンを描画するとき、頂点の順序により指定
  - 背面除去の判定に使用

まとめ

- レンダリング・パイプライン
  - Zバッファ法によるポリゴン描画の仕組み
  - 座標変換とラスタライズ
- 座標変換
- ラスタライズ
- OpenGLでのレンダリング設定
- OpenGLでのポリゴン描画

次回予告

- OpenGL演習
  - ポリゴンモデルの描画