コンピュータグラフィックス特論 II

第1回 コンピュータグラフィックスの基礎

九州工業大学 尾下 真樹

講義担当

- 尾下 真樹（おした まさき）
  - 居室：研究棟 W623
  - e-mail: oshita@ces.kyutech.ac.jp
  - http://www.cg.ces.kyutech.ac.jp
  - 研究内容
    - コンピュータアニメーション技術の研究
      - 仮想人間の動作生成・制御、操作インターフェース、物理シミュレーション、アニメーション制作システム

本日の内容

- ガイダンス
- コンピュータグラフィックスの概要と応用
- 3次元グラフィックスの要素技術
- 3次元グラフィックスのプログラミング
- 演習問題

本講義の目標

- 最近の3次元CGで使われている応用技術について学ぶ
- 学習した技術を実際に応用して、3次元CGを使ったプログラムを作成できるようになる
- コンピュータグラフィックス技術を用いるソフトウェアを作成するときに必要となる、実用的な技術を中心に学習する

本講義の内容

- 本講義で学ぶ応用技術
  - 視点操作
  - 幾何形状データの読み込み
  - 影の表現（高度な描画技術）
  - キーフレームアニメーション
  - 物理シミュレーション
  - 衝突判定
  - キャラクタアニメーション

→ 特に（キャラクタ）アニメーション関連の技術を重点的に扱う

本講義の意義

- 本大学院の学生で、コンピュータグラフィックスを専門とする研究・仕事に就く人は少ない？
  - ゲームプログラマ、アニメーション制作者など
  - 本講義の内容は、これらの仕事に直接関係する
- 他の多くのソフトウェアでも、コンピュータグラフィックスが要素技術として使われる
  - ロボットのシミュレーション結果の表示
  - 仮想空間や仮想人間のアニメーション
  - 3次元物体のシミュレーション結果の可視化、等

本講義の意義（続き）

- 研究や仕事で、コンピュータグラフィックスを使ったプログラムを開発する機会があれば、本講義で扱う内容は役に立つ

コースとモジュール

- 本講義は「グラフィックスと応用」モジュールに属する
  - 本モジュールに属する他の講義
    - マルチメディア表現／工学特論（乃万先生）
    - ヒューマンインタフェース（大橋先生）
    - 仮想空間論（砥崎先生）
    - コンピュータビジョン特論II（岡部先生）

    （5科目中3科目の単位を取得すれば、モジュール修了）
- 「グラフィックスと応用」モジュールは、「メディア処理」コースに属する
  - 他のモジュールも修了すれば、コース修了認定

想定する受講者

- コンピュータグラフィックの基礎を理解していること
  - レンダリングの仕組み、座標変換などの考え方
    （学部のCG科目や大学院のCG特論I を履修している）
  - OpenGL を使ったプログラミング
  - 1・2回目の授業でこれらの基礎知識についても少し復習
- C/C++を使ったプログラミングができること
  - 標準的な学部4年生程度のプログラミング能力
  - 具体的なアルゴリズムが与えられれば、自力でプログラムを作成できる程度の能力はあること
  - 本授業のプログラミング演習で必要となる、やや高度なプログラミング技術も、本授業で扱う

For International Students

- This class covers advanced techniques on computer graphics and animation. The students can learn practical techniques through lectures and programming exercises.
- **This class is taught in Japanese**. Although some materials have English version, most of materials are Japanese only. Reports in Japanese or English are acceptable. A foreign student who are not so fluent in Japanese can still take this class.
- The students must have fundamental programming skills of C++. Basic knowledge on computer graphics and OpenGL programming are not mandatory but desirable.

講義の進め方

- 基本的に PowerPointの資料を使って講義
  - 講義スライドは事前に公開
    - 印刷したものを配布はしないので、必要な人は各自印刷して来ること
  - 教科書はなし
    - 参考文献は各テーマごとに適宜紹介する
- 授業時間中は、講義のみで、プログラミング演習は行わない
  - プログラミング演習は授業時間後に各自で行う

講義資料

- 講義資料は、授業のウェブページで公開
  - 遅くとも、授業の前日の正午までには公開予定
  - 必要な人は各自印刷すること
- 演習・レポート課題に関する資料も本ページで公開
- レポート提出方法については別途指示

授業のウェブページ：
http://www.cg.ces.kyutech.ac.jp/lecture/cg2/

成績評価

- プログラミング演習課題レポート(80%)
  - プログラム作成の課題を出し、そのレポートの内容により評価
    - 5回程度のレポート課題を出す
- 出席状況・演習問題(20%)
  - 毎回の授業に出席して講義内容を理解する
  - 講義中に演習問題(ミニテスト)を行う
    - 学会等での欠席は事前に欠席届を出せば考慮(ただし演習問題の点数はなし)、就職活動は欠席扱い

レポート課題

- 各テーマごとに、授業で学習した技術を応用したプログラミング課題のレポートを出す
- レポート課題の内容
  - 元になるサンプルプログラムを、授業のウェブページで公開
    - 大部分のプログラムはあらかじめ作成済みなので、各自で作成する必要はない
  - 重要な部分の処理のみを、各自で作成
  - 作成したプログラムを提出
    - 文章でのレポートの作成・提出は不要

講義予定

- 基礎技術(全2回)
  - コンピュータグラフィックスの基礎
  - OpenGLプログラミングの基礎
- 応用技術(全13回)
  - 視点操作
  - 幾何形状データの読み込み
  - 影の表現(高度な描画技術)
  - キーフレームアニメーション
  - 物理シミュレーション
  - 衝突判定
  - キャラクタアニメーション

コンピュータグラフィックスの概要と応用

コンピュータ・グラフィックス

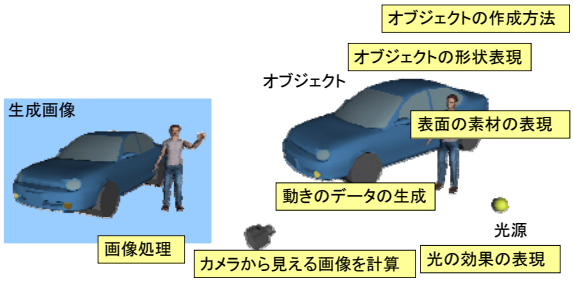
- 広い意味でのコンピュータ・グラフィックス
  - コンピュータを使って視覚データを扱う技術
  - 画像・動画、ユーザインターフェースなど、2次元グラフィックスを含む
- 狭い意味でのコンピュータ・グラフィックス
  - 3次元の形状・空間データを扱い、計算によって画像を生成する技術
    - ただし、最終的なアウトプットとしては、2次元の画像データになることが多い
  - この講義では、こちらを主に扱う

グラフィックスの要素技術

- CG画像を生成するためのしくみ
  - 仮想空間にオブジェクトを配置
  - 仮想的なカメラから見える映像を計算で生成
  - オブジェクトやカメラを動かすことでアニメーション

グラフィックスの要素技術

- コンピュータグラフィックスの主な技術



グラフィックス応用の分類

- オフライン画像生成
  - 静止画、映画など
  - アニメータらが多くの時間をかけて制作
- オンライン画像生成
  - コンピュータゲーム、シミュレーション、VR など
  - ユーザの操作などに応じて、インタラクティブに画像を生成する必要がある
    - ただし、物体の形状データやモーションデータは、前もって作成されていることが多い

グラフィックス応用による違い

- 基本的な原理や考え方は共通
- オフライン画像生成のための技術
  - 多少時間がかかっても良いので、高画質の画像を生成する必要がある
  - 基本的には既存のソフトウェアが利用可能なので、各技術の概要を大体知っておけば実用には十分
- オンライン画像生成のための技術
  - 多少画質を犠牲にしても、リアルタイムに画像を生成する必要がある
  - 自分でプログラムを作成しなければならないことが多いため、各技術を具体的に理解しておく必要がある
  - 本講義では、こちらの応用のための技術を主に扱う

市販ソフトウェアの利用

- 市販のアニメーション・ソフトウェアが広く使われている
  - Maya, Softimage, 3ds max, LightWave など
- 既存のソフトウェアがサポートする技術は、自分でわざわざプログラムを作成しなくて良い場合もある
  - 形状データやモーションデータの作成など
  - うまく組み合わせることが重要

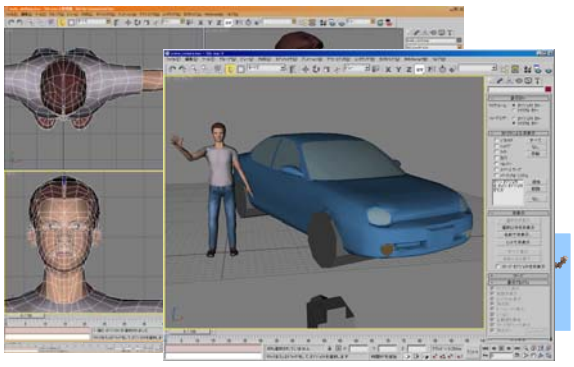
CGプログラミング

- 市販ソフトウェアと組み合わせたプログラムの例
  - 市販ソフトウェアを使い、製品の形状データをモデリング
    → 自作のプログラムで有限要素法シミュレーションを行い、解析結果をCGで描画
  - 市販ソフトウェアを使い、キャラクタの骨格・形状データやモーションデータを作成
    → 自作のプログラムでは、上記のデータを読み込み、ユーザの操作に応じてモーションを再生
- オンライン・アプリケーションでは、描画処理や動きの再生などは、自分で作成する必要がある
  - 最近は、既存のミドルウェアやゲームエンジン（Unity、Unreal等）を用いる場合もある

市販ソフトウェアの利用

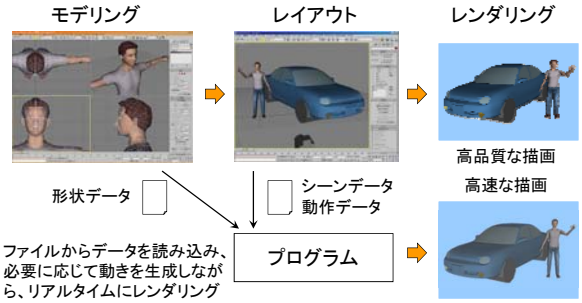
- 一般的なCG作品制作の流れ
  - モデリング・・・各物体ごとの形状データを作成
  - レイアウト・・・空間に物体を配置、動きを設定
  - レンダリング・・・静止画像orアニメーション生成

市販ソフトウェアの利用



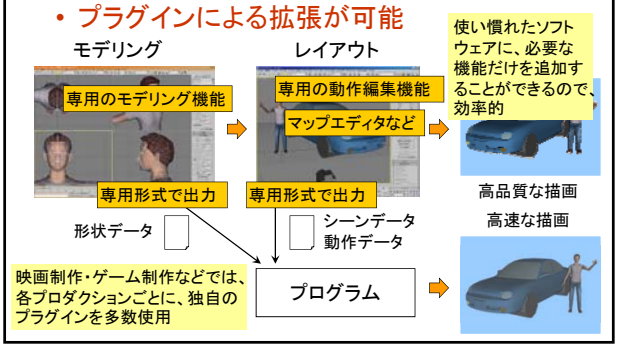
市販ソフトウェアの利用

- 既存ソフトウェアと組み合わせたプログラミング



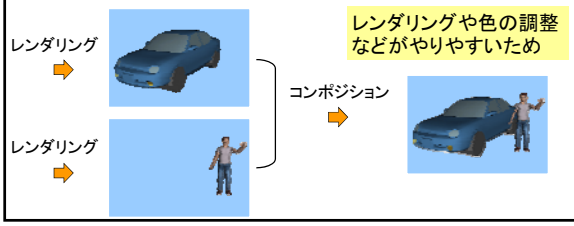
市販ソフトウェアの利用

- プラグインによる拡張が可能



オフライン・アニメーション

- コンポジションも重要な作業のひとつ
  - 実際の制作では、各オブジェクトや効果を個別にレンダリングして、後でまとめて合成・編集することが多い（専用の編集ソフトを使用）



コンポジションの利用

- 映画などでは、フルCGよりも、実写＋CGの合成が多い
  - 実写では実現できないような映像のみをCGで表現
  - 我々の身近にある物、特に人間などはCGで再現することが難しい
    - 少しでも不自然なところがあるとすぐに目立つ
  - あまり身近にないような物、実写では絶対に撮影できないような物をCGで作り出す
  - 実写で撮影可能なものにはCGは使わず、実写とCGを合成

3Dグラフィックスと実写の関係

- 3Dグラフィックス
  - 制作には労力がかかる
  - 存在しないものも表現可
  - 人間などの再現は難しい
- 実写
  - 実物をそのまま撮影できる
  - 人間などは実写の方が向いている
- 両者をうまく使い分けて撮影・生成し、最終的に合成して映像を作る方法が一般的

Jurassic Park III
Universal Pictures

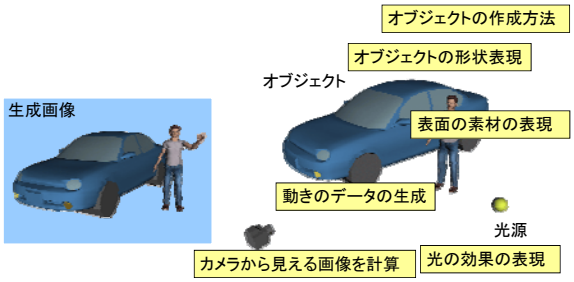
3次元グラフィックスの要素技術

3次元グラフィックスのしくみ（復習）

- CG画像を生成するための方法
  - 仮想空間にオブジェクトを配置
  - 仮想的なカメラから見える映像を計算で生成
  - オブジェクトやカメラを動かすことでアニメーション

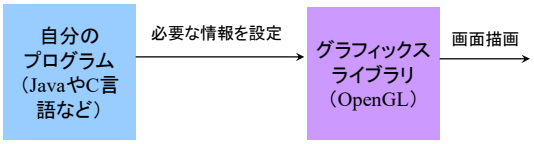
3次元グラフィックスの要素技術

- コンピュータグラフィックスの主な技術



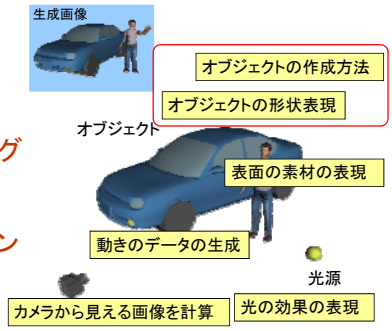
グラフィックスライブラリの利用

- グラフィックスライブラリ（OpenGLなど）
  - 要素技術を簡単に利用できる
  - 要素技術の仕組みは理解する必要がある
  - 詳しくは、本講義の演習で説明（次回以降）



モデリング

- モデリング
- レンダリング
- 座標変換
- シェーディング
- マッピング
- アニメーション



モデリング

- モデリング（Modeling）
  - コンピュータ上で、物体の形のデータを扱うための技術
  - 形状の種類や用途によって、さまざまな表現方法がある
  - 形状データの表現方法だけでなく、どのようにしてデータを作るかという、作成方法も重要になる

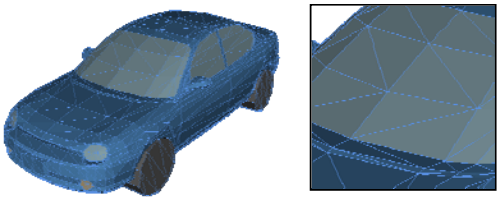
各種モデリング技術

- サーフェスモデル
  - ポリゴンモデル
  - 曲面パッチ
  - サブディビジョンサーフェス
- ソリッドモデル
  - 境界表現
  - CSGモデル
- その他のモデル

※ 詳しくは、後日の講義で説明

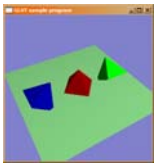
ポリゴンモデル

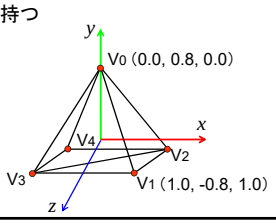
- 物体の表面の形状を、多角形(ポリゴン)の集まりによって表現する方法
  - 最も一般的なモデリング技術
  - 本講義の演習でも、ポリゴンモデルを扱う



ポリゴンモデルの表現例

- 四角すいの例
  - 5個の頂点 と 6枚の三角面(ポリゴン) によって表現できる
    - 各三角面は、どの頂点により構成されるかという情報を持つ





三角面
{ V0, V3, V1 }
{ V0, V2, V4 }
{ V0, V1, V2 }
{ V0, V4, V3 }
{ V1, V3, V2 }
{ V4, V2, V3 }

ポリゴンモデルの表現例(続き)

- プログラムでの表現例(配列による表現)
  - 頂点座標の配列
  - ポリゴンを構成する頂点番号の配列

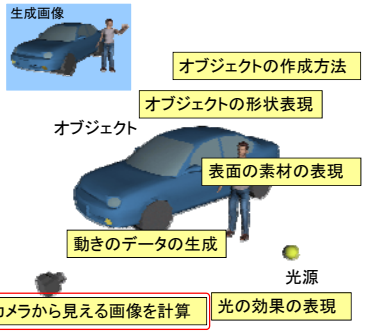
```
const int  num_pyramid_vertices = 5;    // 頂点数
const int  num_pyramid_triangles = 6;   // 三角面数

// 角すいの頂点座標の配列
float  pyramid_vertices[ num_pyramid_vertices ][ 3 ] = {
          { 0.0, 1.0, 0.0 }, { 1.0,-0.8, 1.0 }, { 1.0,-0.8,-1.0 },
          {-1.0,-0.8, 1.0 }, {-1.0,-0.8,-1.0 }
};

// 三角面インデックス(各三角面を構成する頂点の頂点番号)の配列
int  pyramid_tri_index[ num_pyramid_triangles ][ 3 ] = {
          { 0,3,1 }, { 0,2,4 }, { 0,1,2 }, { 0,4,3 }, { 1,3,2 }, { 4,2,3 }
};
```

モデリングのまとめ

- コンピュータ上で、物体の形のデータを扱うための技術
- さまざまなモデリングの方法がある
- ポリゴンモデルが一般的
  - 多角形(面)の集まりで形状を表す
- ポリゴンモデルは、配列などの形で、プログラムで表現することができる

レンダリング

- モデリング
- レンダリング
- 座標変換
- シェーディング
- マッピング
- アニメーション

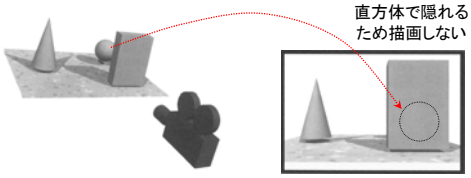


レンダリング

- 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

レンダリングの重要なポイント

- 隠面消去をどのようにして実現するか？
  - 見えるはずのない範囲を描画しない処理
  - 普通に存在する面を全て描いたら、見えるはずのない面まで表示されてしまう



参考書「コンピュータグラフィックスの基礎知識」図2-21

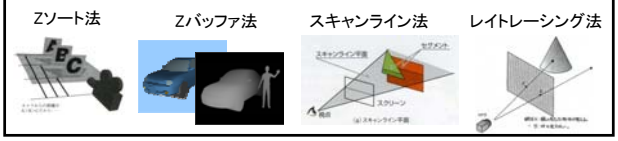
各種レンダリング手法

- 主なレンダリング手法
  - 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

低画質、高速度

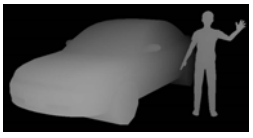
高画質、低速度

  - 隠面消去の実現方法が異なる



Zバッファ法

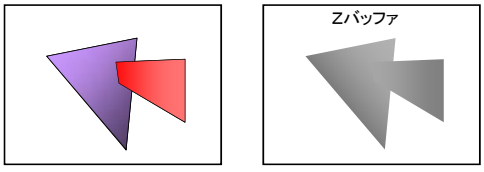
- Zバッファ法
  - 描画を行う画像とは別に、画像の各ピクセルの奥行き情報を持つ Zバッファ を使用する
  - コンピュータゲームなどのリアルタイム描画で、最も一般的な方法（本講義の演習でも使用）



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

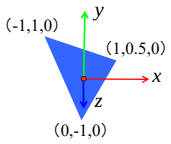
- Zバッファ法による面の描画の例
  - 面を描画するとき、各ピクセルの奥行き値（カメラからの距離）を計算して、Zバッファに描画
  - 同じ場所に別の面を描画するときは、すでに描画されている面より手前のピクセルのみを描画



プログラムの例

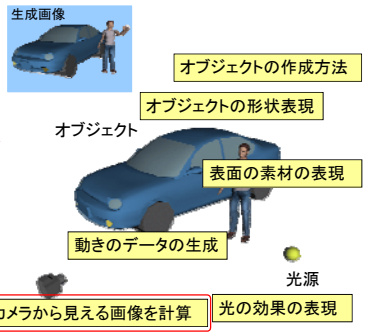
- 1枚の三角形を描画するプログラムの例
  - 各頂点の頂点座標、法線、色を指定して描画

```
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();
```



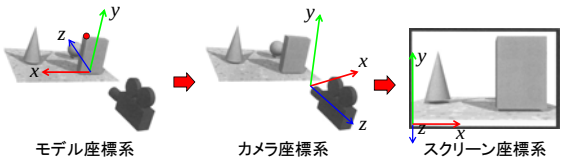
座標変換

- モデリング
- レンダリング
- 座標変換
- シェーディング
- マッピング
- アニメーション



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



アフィン変換

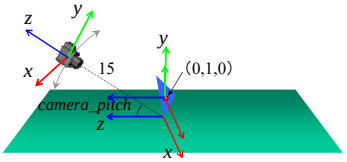
- アフィン変換（同次座標系変換）
  - 4×4行列の演算によって、3次元空間における回転・平行移動・拡大縮小などの処理を計算
  - 同次座標系
    - (x, y, z, w)の4次元座標値によって扱う
    - 3次元座標値は$(x/w, y/w, z/w)$で計算（通常は $w = 1$）

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

  - 非常に重要な考え方（詳しくは後日の講義で説明）

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

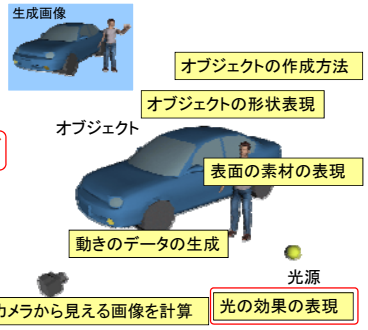
- 適切な変換行列を設定することで、カメラや物体の位置・向きを自在に変更できる

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

```
// 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 );
```

シェーディング

- モデリング
- レンダリング
- 座標変換
- シェーディング
- マッピング
- アニメーション

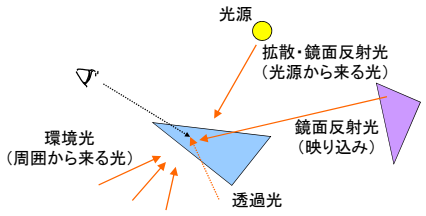


シェーディング

- シェーディング（Shading）
  - 光による効果を考慮して、物体を描く色を決めるための技術
    - 現実世界では、同じ素材の物体でも、光の当たり方によって見え方は異なる
    - コンピュータグラフィックスでも、このような効果を再現する必要がある

光のモデル

- 光を影響をいくつかの要素に分けて計算
  - 局所照明（光源からの拡散・鏡面反射光）
  - 大域照明（環境光、映り込み、透過光）



シェーディングの効果の例

- 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

シェーディングの効果の例

- 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

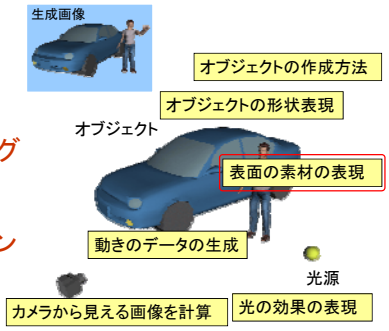
プログラムの例

- 光源の位置や色を設定すると、OpenGL が自動的に光の効果を計算

```
float  light0_position[] = { 10.0, 10.0, 10.0, 1.0 };
float  light0_diffuse[] = { 0.8, 0.8, 0.8, 1.0 };
float  light0_specular[] = { 1.0, 1.0, 1.0, 1.0 };
float  light0_ambient[] = { 0.1, 0.1, 0.1, 1.0 };
glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
glLightfv( GL_LIGHT0, GL_DIFFUSE, light0_diffuse );
glLightfv( GL_LIGHT0, GL_SPECULAR, light0_specular );
glLightfv( GL_LIGHT0, GL_AMBIENT, light0_ambient );
glEnable( GL_LIGHT0 );
glEnalbe( GL_LIGHTING );
```

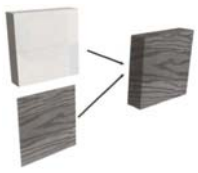
マッピング

- モデリング
- レンダリング
- 座標変換
- シェーディング
- マッピング
- アニメーション

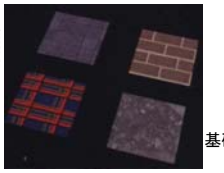


マッピング

- マッピング（Mapping）
  - 物体を描画する時に、表面に画像を貼り付けて描画する技術
  - 複雑な形状データを作成することなく、細かい模様などを表現できる



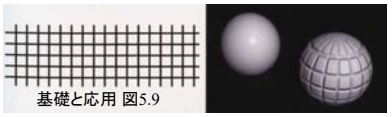
基礎知識 図3-19



基礎と応用 図5.2

高度なマッピング

- 凹凸のマッピング（バンプマッピング）



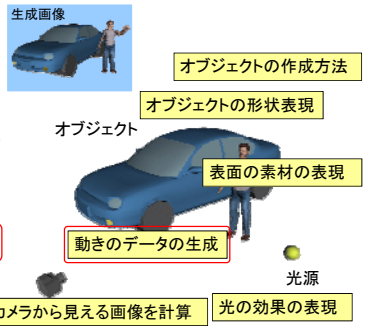
基礎と応用 図5.9

- 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

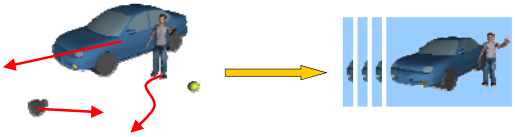
アニメーション

- モデリング
- レンダリング
- 座標変換
- シェーディング
- マッピング
- アニメーション



アニメーション

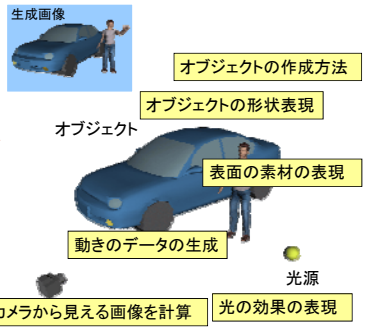
- 動きのデータをもとに、アニメーションを生成



- 対象や動きの種類に応じてさまざまな動きの作成方法がある
  - キーフレームアニメーション、物理シミュレーション、モーションキャプチャ、など

3次元グラフィックスの要素技術

- モデリング
- レンダリング
- 座標変換
- シェーディング
- マッピング
- アニメーション



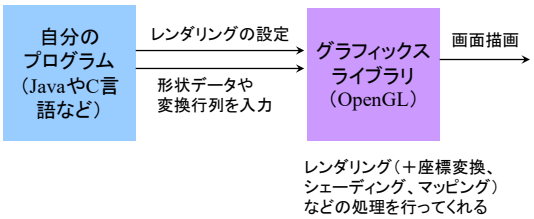
3次元グラフィックスのプログラミング

3次元グラフィックス・プログラミング

- ここまでに説明した技術を実現するようなプログラムを作成することで、3次元グラフィックスを描画できる
  - 全てを自分で実現しようとすると、非常に多くのプログラムを書く必要がある
  - 現在は、OpenGL のような、3次元グラフィックスライブラリが存在するので、これらのライブラリを利用することで、3次元グラフィックスを扱うプログラムを、比較的簡単に作成できる

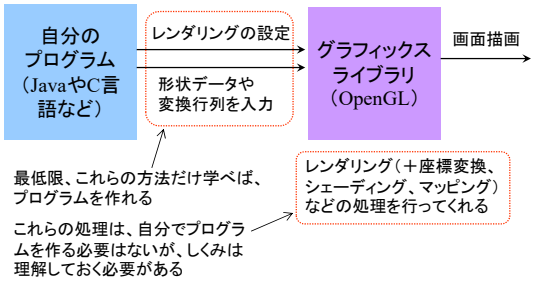
グラフィックスライブラリの利用

- 自分のプログラム と OpenGL の関係



グラフィックスライブラリの利用

- 自分のプログラム と OpenGL の関係



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

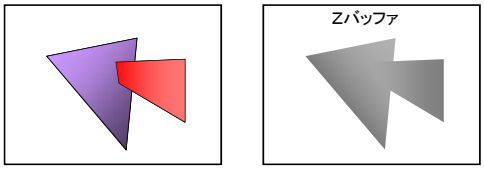
Zバッファ法（復習）

- 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

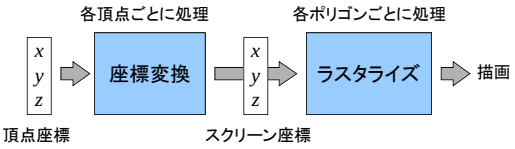
Zバッファの値（手前にあるほど明るく描画）

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

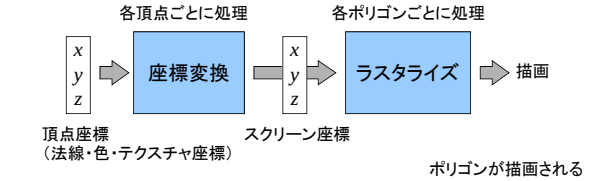
GLUTでのZバッファ法の利用

- 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）

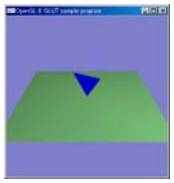


OpenGLにポリゴンの頂点情報を入力

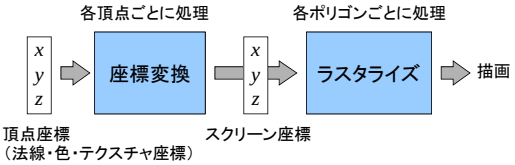
```
glBegin( GL_TRIANGLES );
        glColor3f( 0.0, 0.0, 1.0 );
        glNormal3f( 0.0, 0.0, 1.0 );
        glVertex3f(-1.0, 1.0, 0.0 );
        glVertex3f( 0.0,-1.0, 0.0 );
        glVertex3f( 1.0, 0.5, 0.0 );
glEnd();
```

座標変換 & ラスタライズ

ポリゴンが描画される

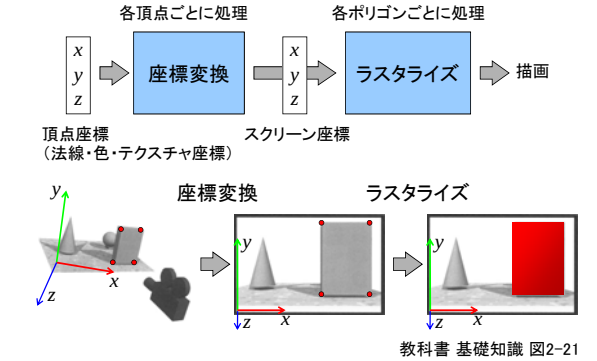


処理の流れ



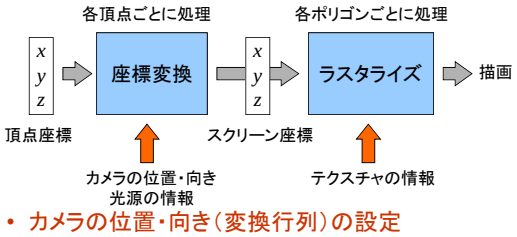
- レンダリング時のデータ処理の流れ
  1. ポリゴンを構成する頂点の座標、法線、色、テクスチャ座標などを入力
  2. スクリーン座標に変換（座標変換）
  3. ポリゴンをスクリーン上に描画（ラスタライズ）

処理の流れ



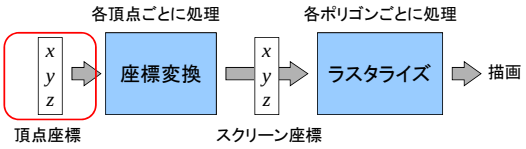
教科書 基礎知識 図2-21

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



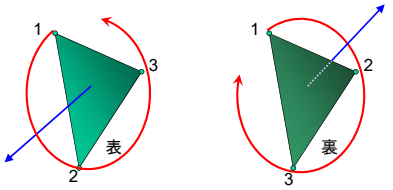
- 物体の情報を入力
  - ポリゴンを構成する頂点の座標・法線・色・テクスチャ座標などを入力
- 表面の素材などを途中で変える場合は、適宜設定を変更

ポリゴンデータ

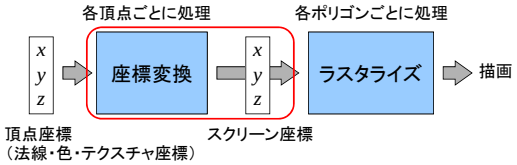
- ポリゴンの持つ情報
  - 各頂点の情報
    - 座標
    - 法線
    - 色
    - テクスチャ座標
      - 法線・テクスチャ座標については、詳細は後日の講義で説明
  - 面の向き
    - 頂点の順番によって面の向きを表す
    - 反時計回りに見える方が表（設定で向きは変更可）

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

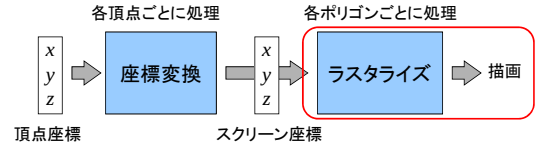


座標変換&ライト計算



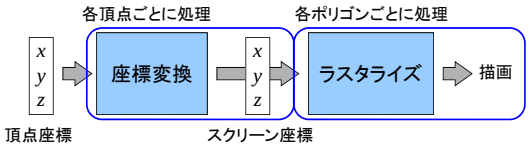
- 座標変換
  - 各頂点のスクリーン座標を計算
  - 法線と光源情報から、頂点の色を計算
    - 色の計算の方法については、後日の講義で説明
  - 面の向きをもとに背面除去、視界外の面も除去

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

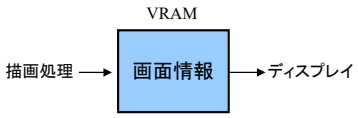
ハードウェアサポート



- 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）

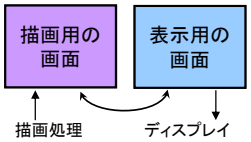
ダブルバッファリング

- 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）

追加資料

- コンピュータグラフィックスの基礎に関するより詳しい内容は、学部の授業の教科書・資料などを参照すること
- システム創成の「コンピュータグラフィックスS」の講義・演習資料は、学部授業用Moodleや尾下研ウェブサイトで公開されている

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（3,600円）
- 「ビジュアル情報処理 －CG・画像処理入門－」
  CG-ARTS協会 編集・出版（2,500円）
- 「3DCGアニメーション」
  栗原恒弥 安生健一 著、技術評論社 出版（2,980円）

まとめ

- ガイダンス
- コンピュータグラフィックスの概要と応用
- 3次元グラフィックスの要素技術
- 3次元グラフィックスのプログラミング
- 演習問題

次回予告

- 第2回　OpenGLプログラミングの基礎

宿題

- 講義のページに置かれているサンプルプログラム（opengl_sample.cpp）をコンパイル、実行して来ること
  - コンパイル環境は、自分の好きな環境を使って構わない（Visual Studio や gcc など）
  - サンプルプログラムの中身も、一通り目を通しておくこと