コンピューターグラフィックスS

第2回　コンピュータグラフィックスの要素技術

システム創成情報工学科　尾下 真樹

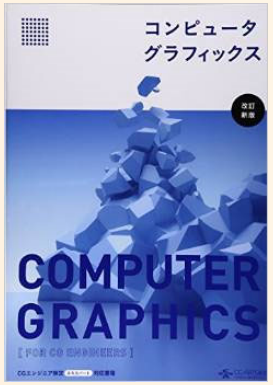
2019年度 Q2

今回の内容

- 前回の復習
- コンピュータグラフィックスの歴史と応用
- 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・プログラミング

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(3,200円)
  - 1～5章 の概要



- 「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)
  - 1～5章 の概要

教科書(参考書)

- 「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)
- 「3次元CGの基礎と応用」
  千葉則茂 土井章男 著、サイエンス社 出版
  (1,900円)
- 「コンピュータグラフィックスの基礎知識」
  塩川厚 著、オーム社 出版 (1,800円)
  - 講義スライドには、一部、これらの参考書から引用した図を使用

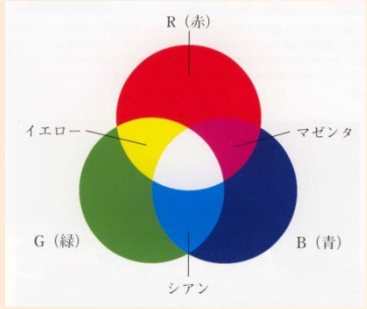
前回の復習

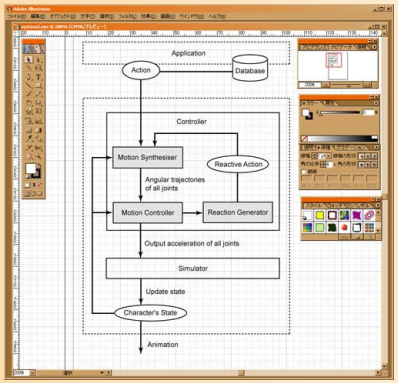
コンピュータグラフィックス

- 「コンピュータグラフィックス」とは？
  - 広い意味では、コンピュータを使って画像・映像を扱う技術の総称
  - 狭い意味では、３次元の形状データをもとに、現実世界のカメラをシミュレートすることによって、画像・映像を生成する技術
    （３次元グラフィックス）

2次元グラフィックス

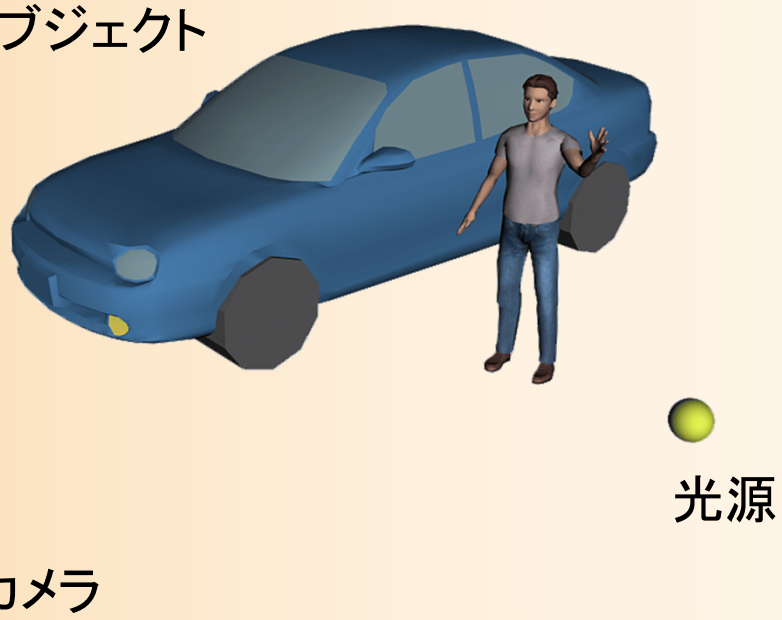
- ピクセルの集まりによる画像の表現
- 光の3原色による色の表現
- 画像の解像度
- 2次元グラフィックスを扱うソフトウェア
  - ペイント系、ドロー系、レタッチ系





3次元グラフィックス

- 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



コンピュータグラフィックスの概要

- コンピュータグラフィックスの主な技術

オブジェクトの作成方法

いかに自然な画像を高速に計算するか

オブジェクトの形状表現

オブジェクト

表面の素材の表現

動きのデータの生成

光源

画像処理

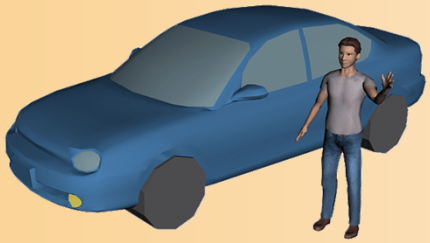
カメラから見える画像を計算

光の効果の表現

コンピュータグラフィックスの分類

- 2次元グラフィックス
  - 画像データ(2次元)を扱う
    - 画像処理、符号化 などの技術

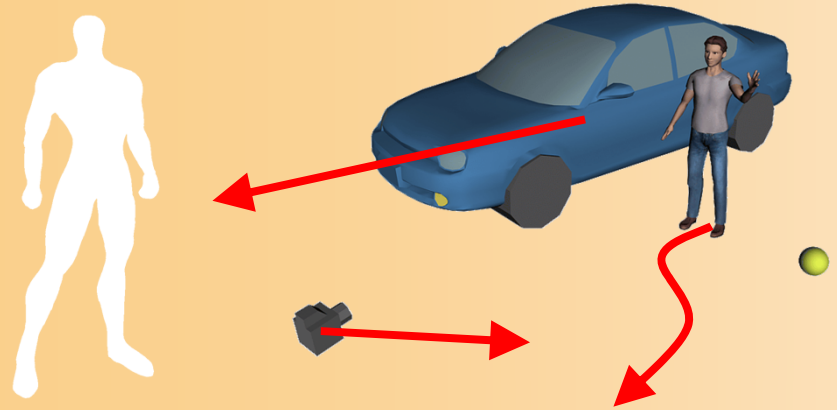
- 3次元グラフィックス
  - シーンデータ(3次元) → 画像データ(2次元)
    - 出力データは、あくまで2次元になることに注意



アニメーション

- ２次元アニメーション
  - 少しずつ変化する画像を
    連続的に生成することで、
    アニメーションになる

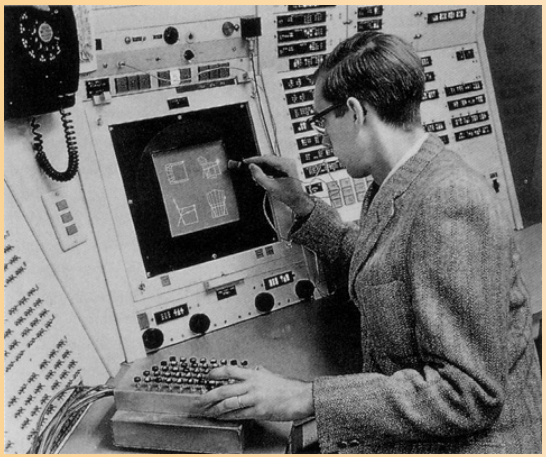
- ３次元アニメーション
  - 動きのデータを与えて、連続画像を生成

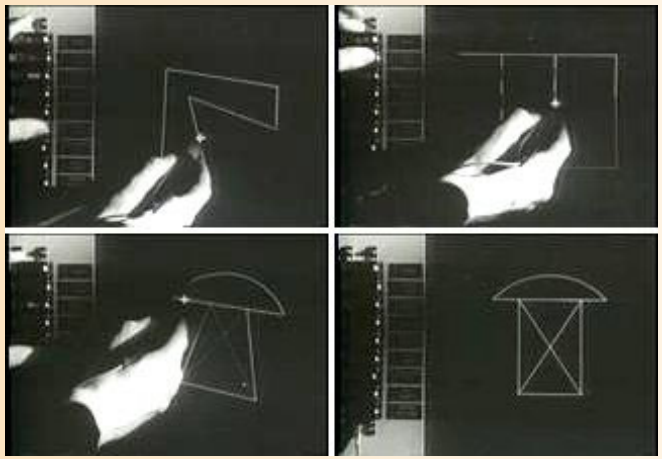


コンピュータグラフィックスの歴史と応用

コンピュータグラフィックスの歴史

- CG研究の始まり（1960年代～）
  - Sutherland が Sketchpadシステムを開発（1963）
    - 世界初のグラフィカルユーザインターフェース





Ivan E. Sutherland

  - 3次元グラフィックス技術の研究が始まる

コンピュータグラフィックスの歴史

- CG研究の始まり（1960年代～）
  - T. Whitted がレイトレーシングを発表（1980）
  - 複数のグループによりラジオシティ法が開発される（1983）
    - レイトレーシング、ラジオシティ法は、光の影響を計算することによって写実的なCGを計算するための手法
  - CG画像・アニメーションが一般に使われ始める

コンピュータグラフィックスの歴史

CGがTVや映画で利用され始める（1980年～）

- 最初は、TV番組のタイトルやアイキャッチの文字などの短いカットにCGが使われる
- 映画への応用
  - ターミネータ2 (1991)、ジュラシックパーク (1993)、タイタニック(1997) 、スターウォーズ(2001) など
- 実写＋CGの合成

Terminator 2, 1991, © Carolco

映画におけるCG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

CGと実写の使い分け

- CG（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

コンピュータグラフィックスの歴史

- フルCG映画の製作（1990年代後半～）
  - 全ての映像をCGだけで製作する試み
    - ToyStory (1995), Shrek (2001), Monsters Inc.(2002)
    - Final Fantasy, The Movie (2001)

Toy Story 2, 1999 © Disney・Pixar

Final Fantasy, 2001 © SQUARE

実写とCGの融合

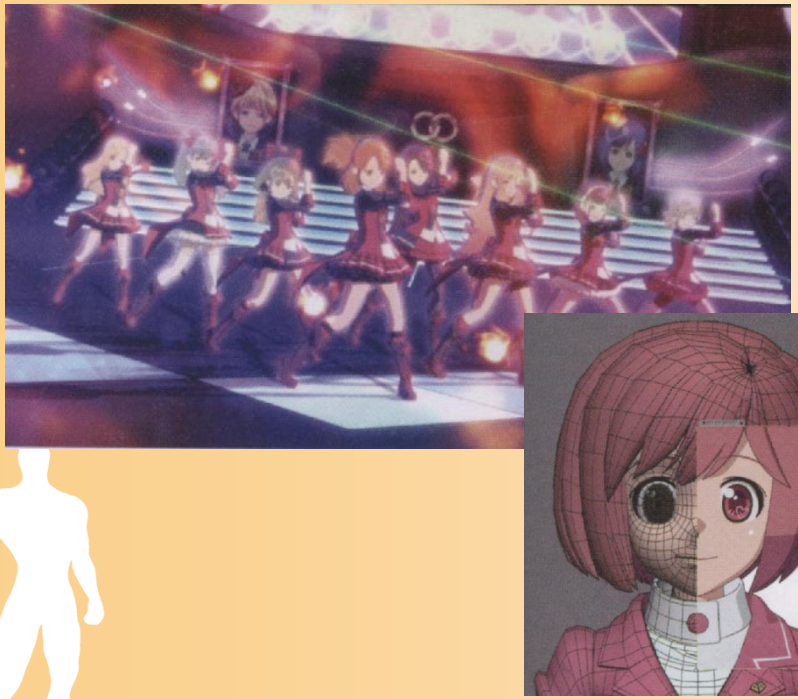
- CGによるリアルな人物の表現（2000年代～）
- イメージベースド・レンダリング（モデリング）
  - 実際の人間のデータを計測し、CGに利用
  - 実写とCGの融合

Spider-Mann 2
Sony Pictures

Matrix
Warner Bros.

セルアニメーションとCGの融合

- ノンフォトリアリスティック・レンダリング
  - 非写実的な手描き風の画像をCGで生成する技術



AKB0048 © サテライト 2013

ジョジョの奇妙な冒険 © 神風動画 2013

CG WORLD 2013年3月号

立体映像

- 右目用･左目用の映像を同時に見せることで立体感を出す
  - 技術的には昔からあったが、最近になって広く普及しつつある

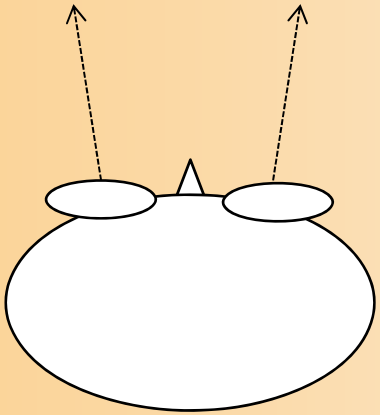
Avatar, 2009 © 20th Century Fox

立体映像

- 右目用･左目用の映像を同時に見せることで立体感を出す

左目用画像

右目用画像



Avatar, 2009 © 20th Century Fox

立体映像

- 右目用・左目用の映像を同時に見せることで立体感を出す
  - 実現方法
    - メガネを使う方法（赤青メガネ、偏光メガネ、シャッターメガネ）
    - メガネが不要な方法
    - ヘッドマウントディスプレイを使う方法
  - 映像の制作方法
    - 2台のカメラで撮影（or CG生成）
    - 映像に擬似的に奥行きを付与

Avatar, 2009 © 20th Century Fox

コンピュータアニメーションの応用

- オフライン・アニメーション
  - 映画やTVなど
  - あらかじめ登場人物の動きが決まっている
- オンライン（リアルタイム）・アニメーション
  - コンピュータゲームなど
  - ユーザの操作に応じてダイナミックに動きを変化させる必要がある
  - 自然な動きを生成するのは非常に難しい

VirtuaFighter, 1993 © SEGA

オンライン・アニメーションの応用

- コンピュータゲーム
- 仮想スタジオ
  - 仮想的なセットの中で撮影
  - 仮想的なキャラクタと実写のキャラクタを合成
- マルチユーザ仮想現実環境
  - ゲーム（Final Fantasy等）
  - コミュニケーション
  - 教育、トレーニング

ウゴウゴルーガ, 1993 © フジTV

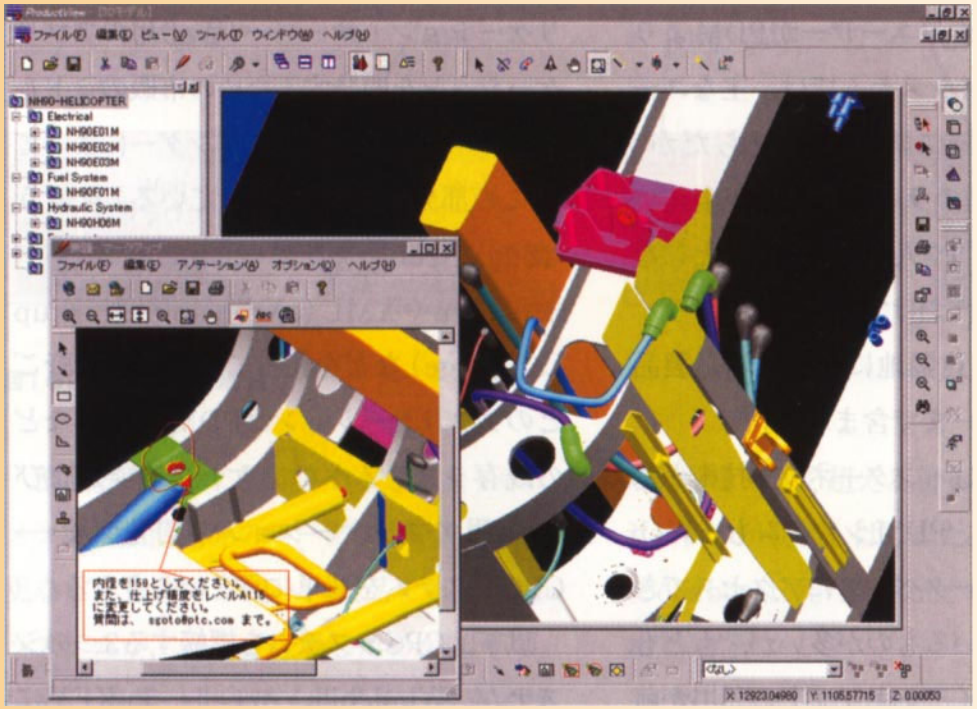
Final Fantasy Online 2002, © SQUARE

コンピュータグラフィックスの応用

- 映画
- コンピュータゲーム
- CAD
- シミュレーション
- 仮想人間(ヴァーチャル・ヒューマン)
- 可視化（ビジュアライゼーション）
- ユーザインターフェース

CAD

- CAD (Computer Aided Design)
  - コンピュータによる設計支援

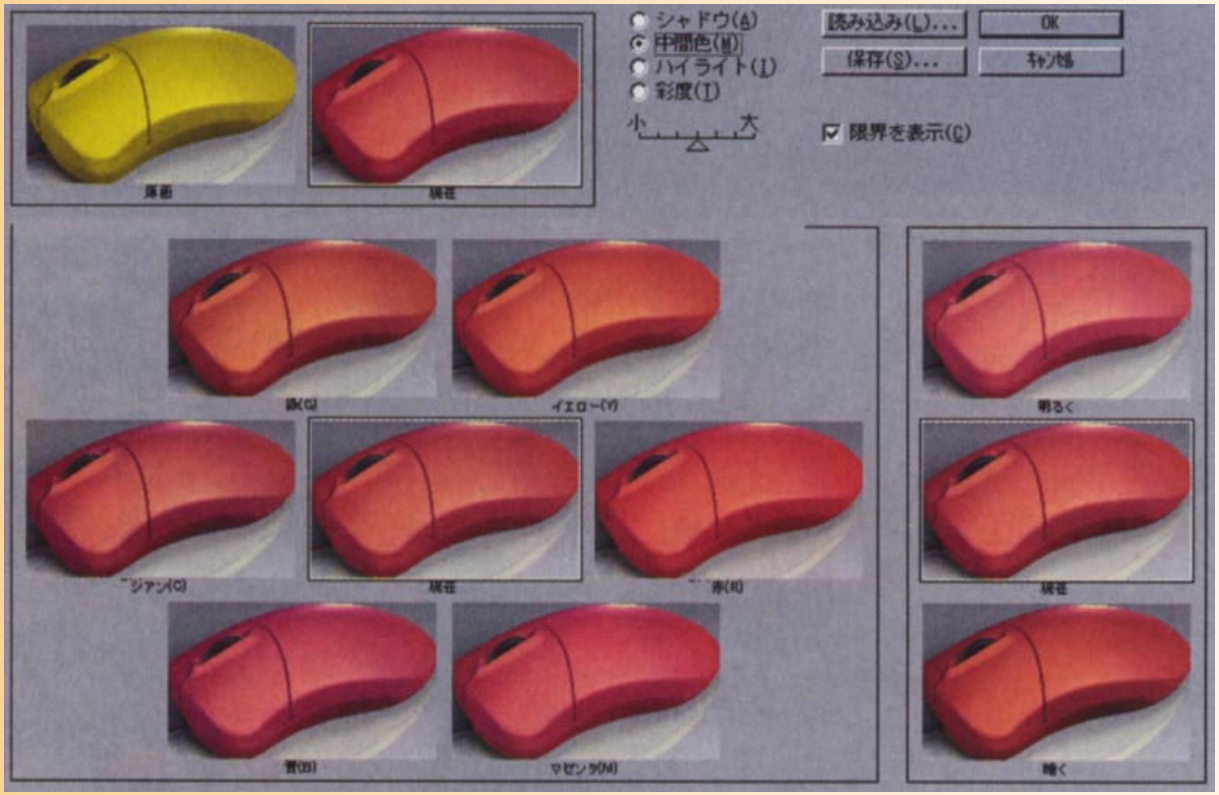


CADの利点

- デザイン作業が簡単
  - 自由にコピー、やり直しなどができる
  - デザイン結果がCGで確認できる
- クライアントにアピールしやすい
  - CGであれば非専門家にも分かりやすい
- データの管理・取り扱いがしやすい
  - CAM (Computer Aided Manufacturing)
    - コンピュータによる製造支援
  - 現在、自動車業界などの製造業ではデータのデジタル化が急速に進められつつある

CADの利点

- いろんなデザインを簡単に比較できる

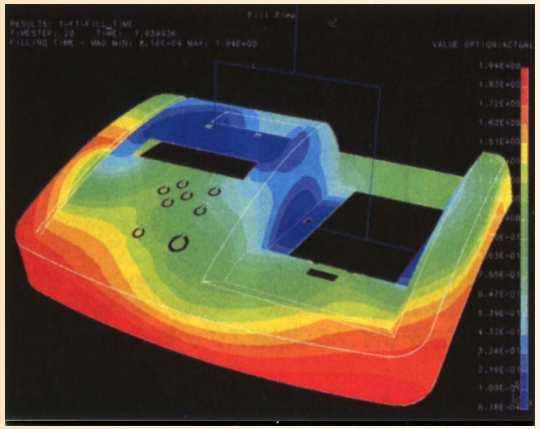


日経CG 2000年7月号 p143

CADの利点

- デザインした製品の性能・使いやすさなどを計算機上で評価できる
  - CAE (Computer Aided Engineering)
    - コンピュータによる技術的・工学的な設計の支援
    - 構造解析、空力・破壊シミュレーションなど

日経CG 1999年10月号 p142



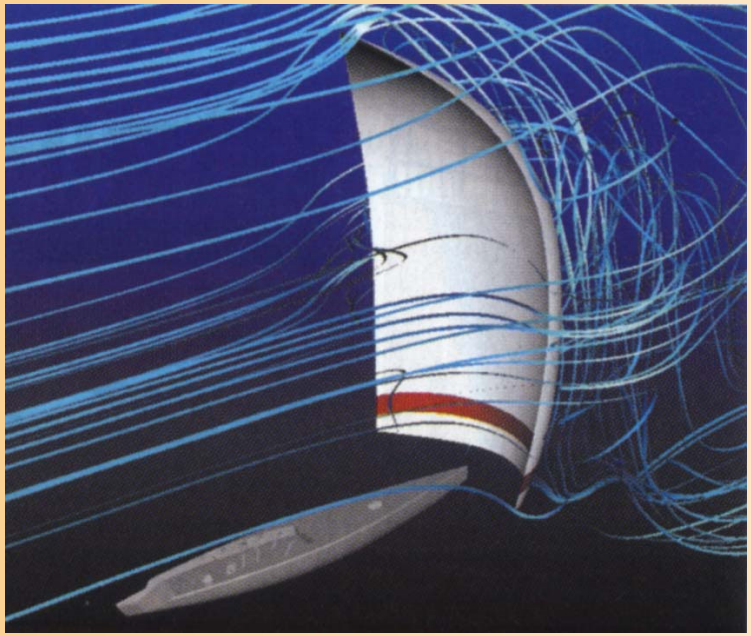
日経CG 2000年2月号 p150

シミュレーション

- シミュレーションはコンピュータグラフィックスと非常に近い技術
  - 現実の物体になるべく近いモデルを計算機上に構築するという点で共通
  - シミュレーションでは、動きのモデルや、動力学的情報（質量・慣性テンソル、摩擦係数）も必要になる
  - シミュレーションの結果を利用者に表示するためにも通常CGが使われる

各種シミュレーション技術

- 車、飛行機などの性能予測
- 衣服デザイン

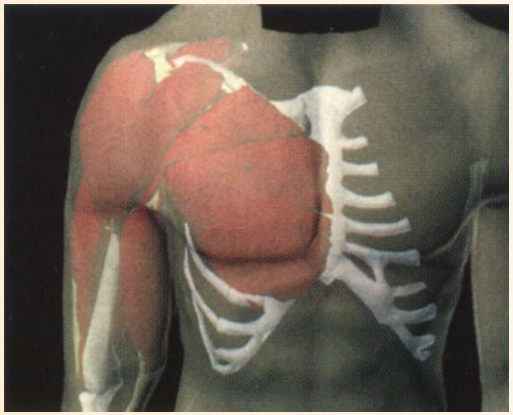


ヨットの空力解析
日経CG 2000年3月号 p117

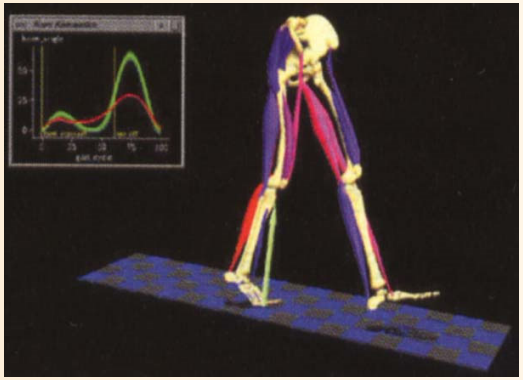
衣服シミュレーション
digital fashion, inc.

ヴァーチャル・ヒューマン

- 仮想的な人間を計算機上に作り出す技術
  - 骨格、筋肉、内臓などもできるだけ正確にモデル化
  - 医学などにも活用
  - アニメーションに応用すれば、よりリアルな皮膚の変形などがシミュレートできる
  - 脳（思考）のモデル化は可能？



Visible Human Project



筋骨シミュレーション SIMM

日経CG 2000年3月号 p120

コンピュータグラフィックスの応用

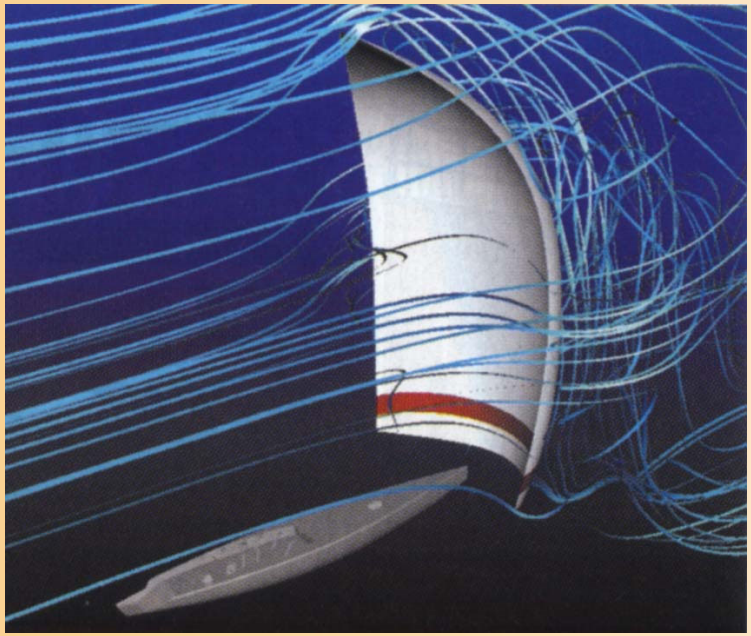
- 映画
- コンピュータゲーム
- CAD
- シミュレーション
- 仮想人間(ヴァーチャル・ヒューマン)
- 可視化（ビジュアライゼーション）
- ユーザインターフェース

可視化（ビジュアライゼーション）

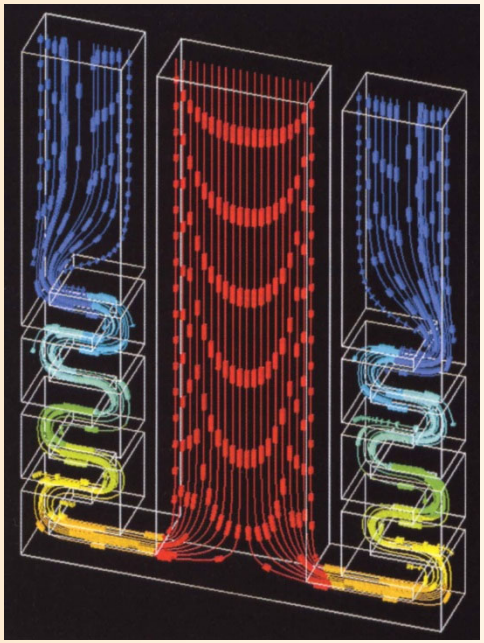
- 人間の目に見えない情報を、CGを使って図形にすることによって、人間が理解しやすくするための技術
- 可視化の技術は大きく2種類に分けられる
  - 科学技術計算の可視化（Scientific Visualization）
    - 3次元空間の実体に関連した情報を可視化
  - 情報可視化（Information Visualization）
    - 比較的最近になって研究されている技術
    - 3次元空間とはあまり関係のない抽象的情報を可視化

科学技術計算結果の可視化

- シミュレーション結果などを可視化
  - 流れ、電界、気象情報など



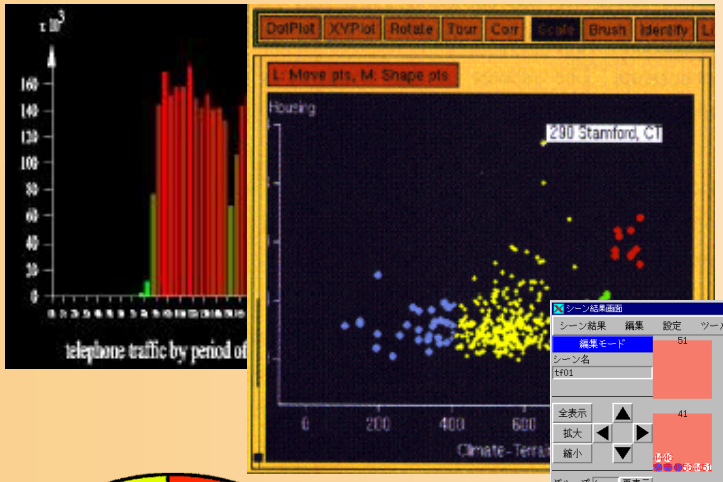
ヨットの空力解析
日経CG 2000年3月号 p117



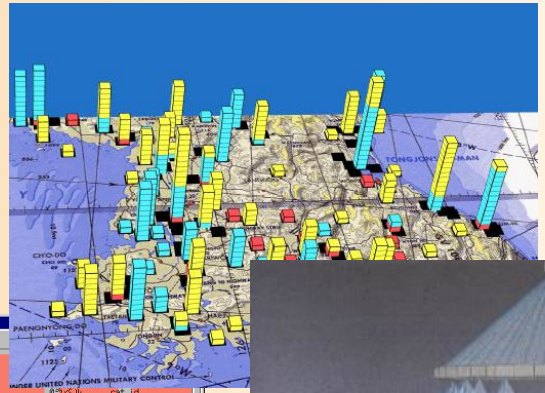
医療機器中の血液の流れ
日経CG 2000年9月号 p177

情報可視化

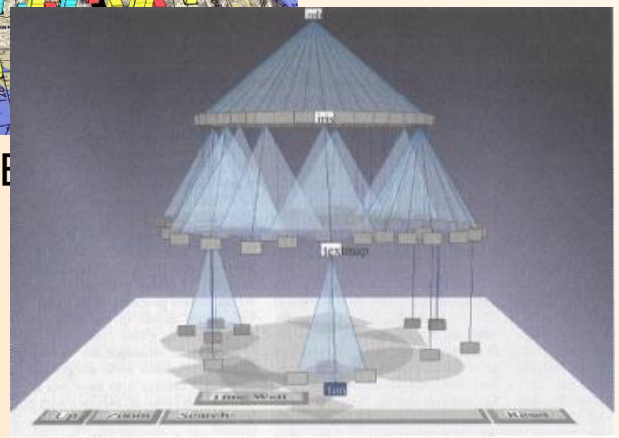
- 大量の情報をいかに分かりやすく見せるか
  - ツリー構造、グラフなどを利用



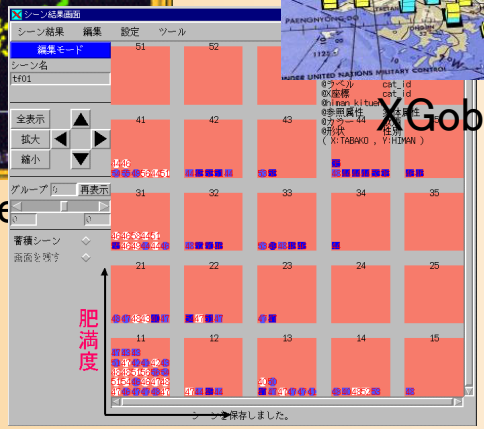
Spotfire



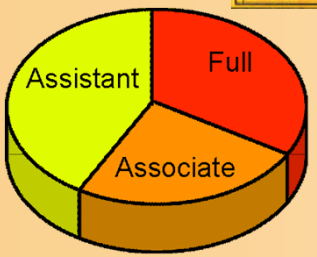
XGobi[



ConeTree [Robertson91]



INFOVISOR[NTT99]

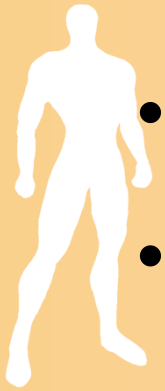


ユーザーインターフェース

- グラフィックスを用いることでユーザとコンピュータの間の情報交換を助ける
  - 計算機の情報を分かりやすくユーザに伝える
  - ユーザの意図をなるべく簡単な操作でコンピュータに伝える
    - 最近は、見た目の美しさにも力が注がれつつある

コンピュータグラフィックスの応用

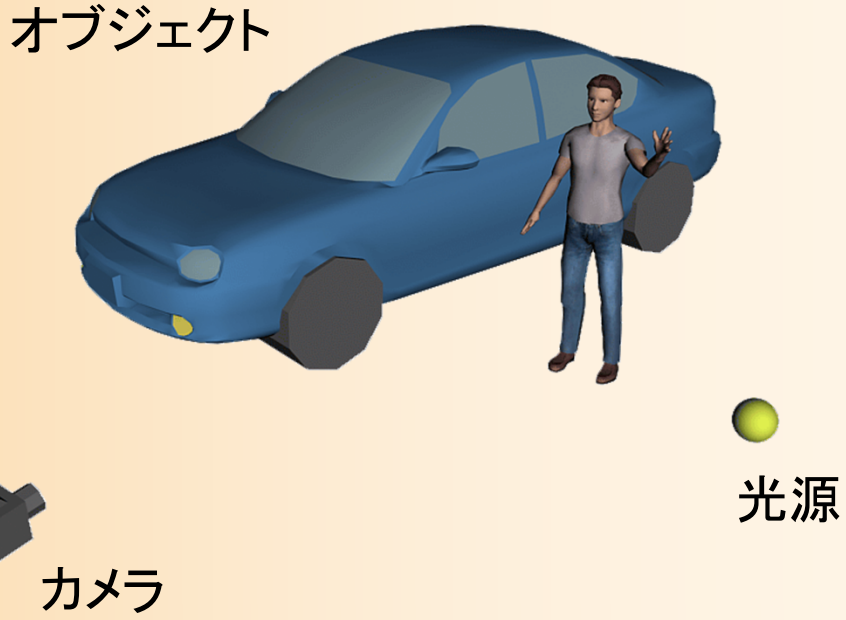
- 映画
- コンピュータゲーム
- CAD
- シミュレーション
- 仮想人間(ヴァーチャル・ヒューマン)
- 可視化（ビジュアライゼーション）
- ユーザインターフェース



3次元グラフィックスの要素技術

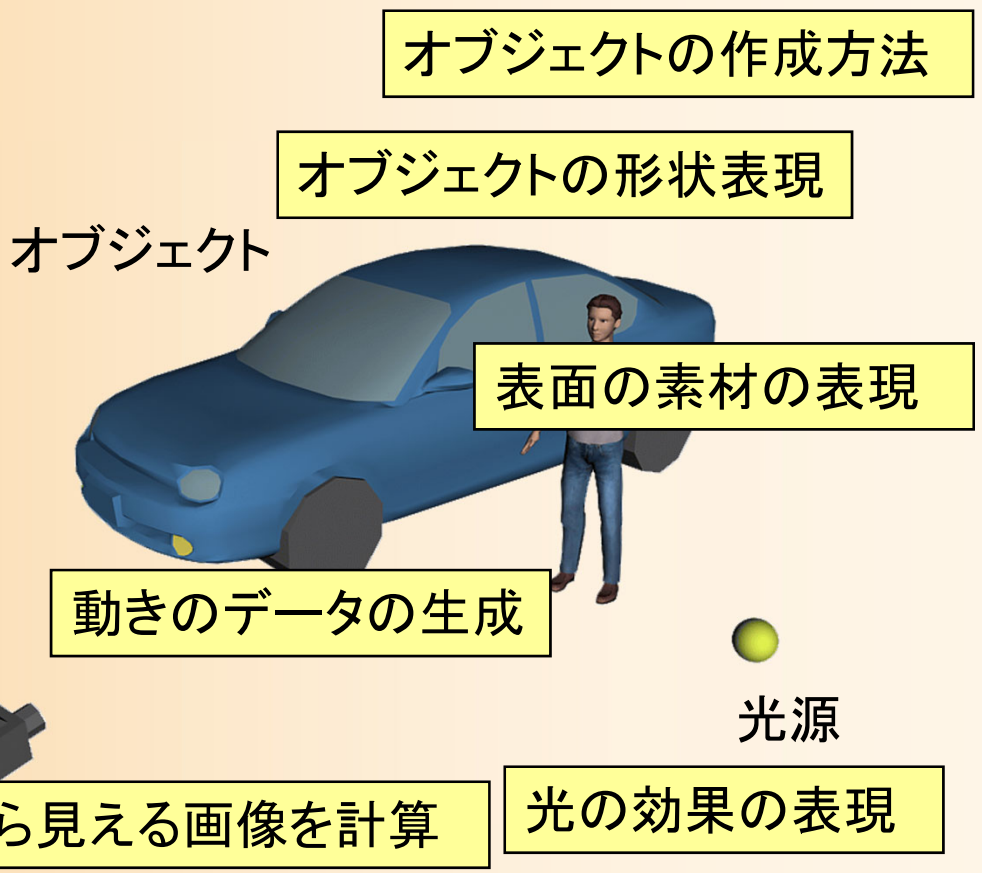
3次元グラフィックスのしくみ（復習）

- 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



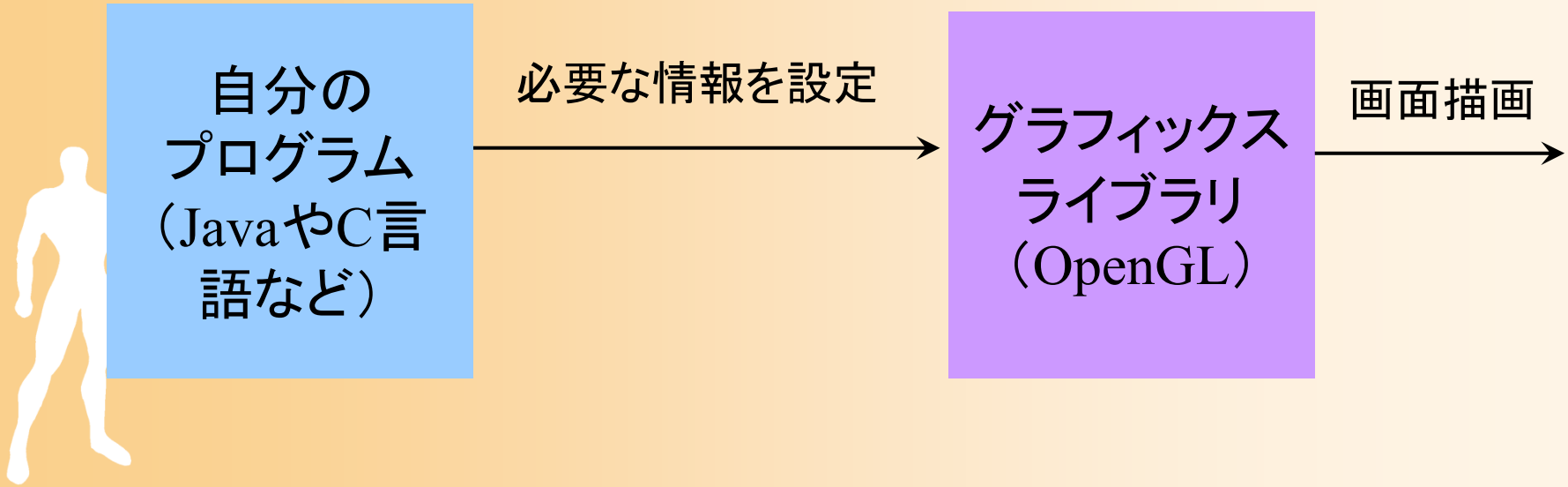
3次元グラフィックスの要素技術

- コンピュータグラフィックスの主な技術



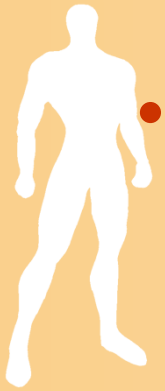
グラフィックスライブラリの利用

- 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



オブジェクトの作成方法

オブジェクトの形状表現

オブジェクト

表面の素材の表現

動きのデータの生成

光源

カメラから見える画像を計算

光の効果の表現

モデリング

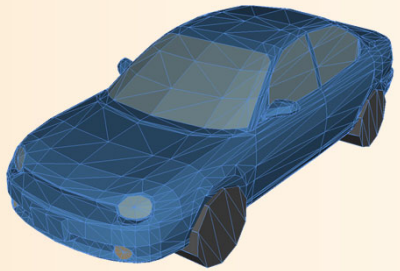
- モデリング（Modeling）
  - コンピュータ上で、物体の形のデータを扱うための技術
  - 形状の種類や用途によって、さまざまな表現方法がある
  - 形状データの表現方法だけでなく、どのようにしてデータを作るかという、作成方法も重要になる

各種モデリング技術

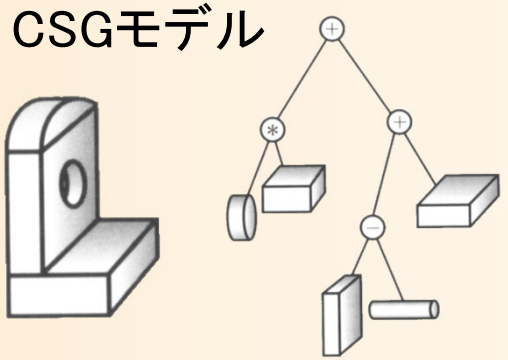
- サーフェスモデル
  - ポリゴンモデル
  - 曲面パッチ
  - サブディビジョンサーフェス
- ソリッドモデル
  - 境界表現
  - CSGモデル
- その他のモデル

※ 詳しくは、後日の講義で説明

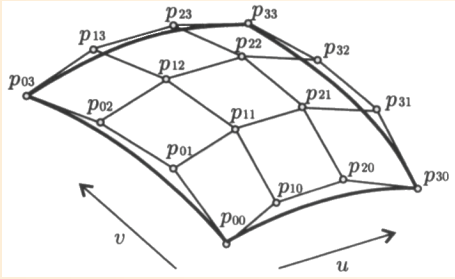
ポリゴンモデル



CSGモデル

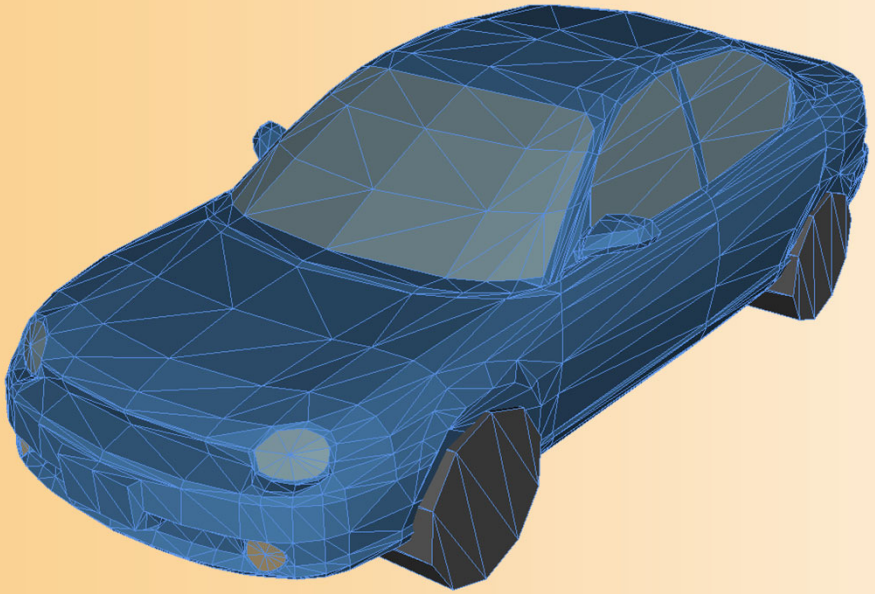


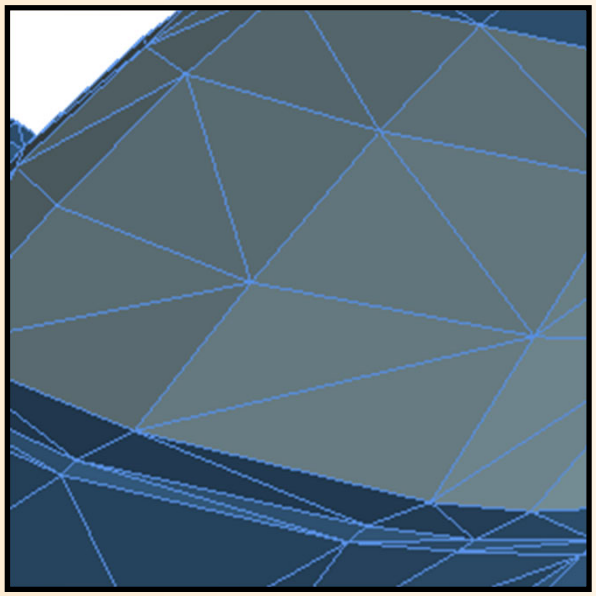
曲面パッチ



ポリゴンモデル

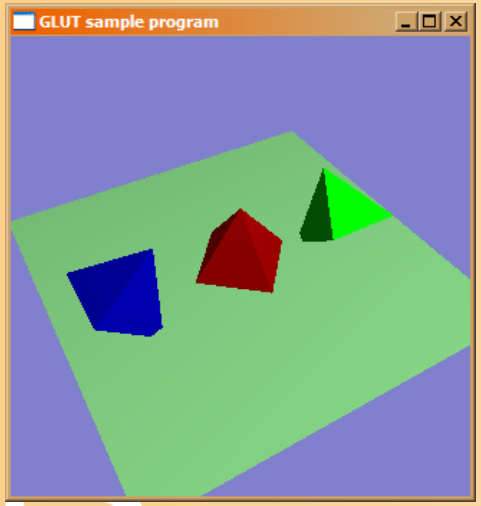
- 物体の表面の形状を、多角形（ポリゴン）の集まりによって表現する方法
  - 最も一般的なモデリング技術
  - 本講義の演習でも、ポリゴンモデルを扱う

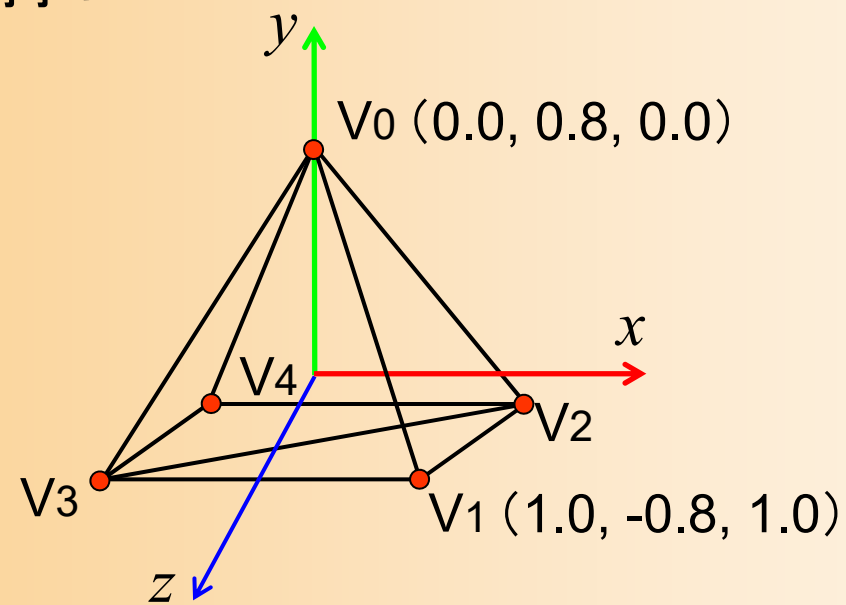




ポリゴンモデルの表現例

- 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

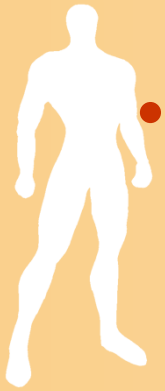
```
const int  num_pyramid_vertices = 5;    // 頂点数
const int  num_pyramid_triangles = 6;   // 三角面数

// 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

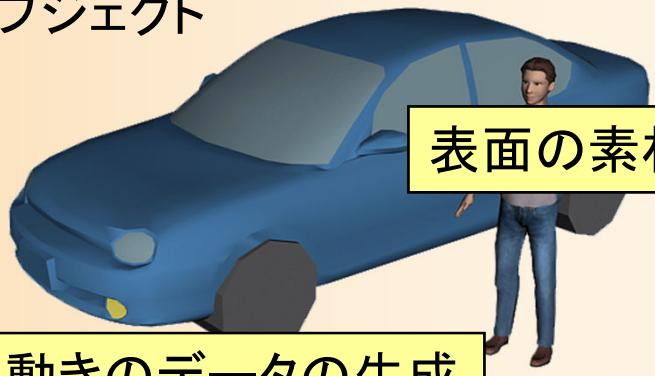
- モデリング
- レンダリング
- 座標変換
- シェーディング
- マッピング
- アニメーション



オブジェクトの作成方法

オブジェクトの形状表現

オブジェクト



表面の素材の表現

動きのデータの生成

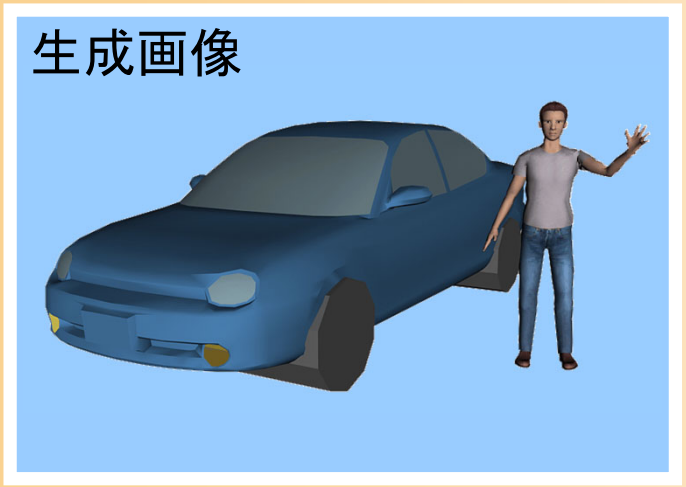
光源

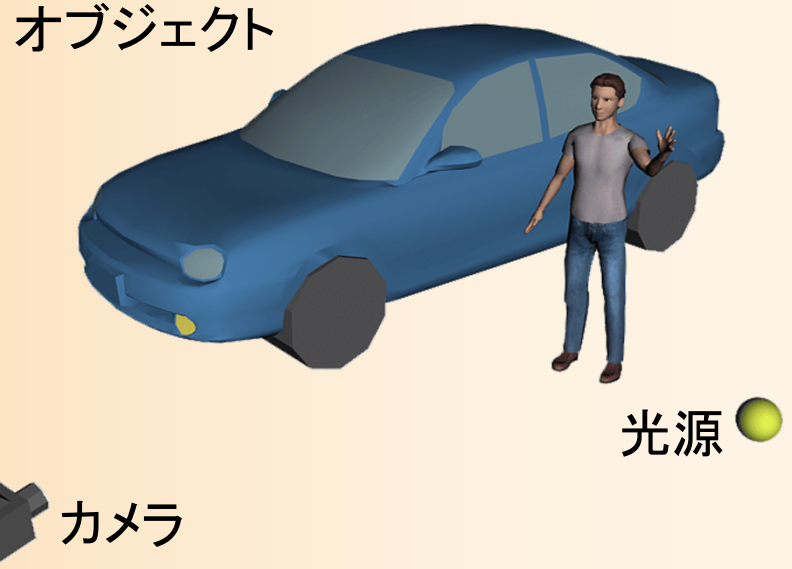
カメラから見える画像を計算

光の効果の表現

レンダリング

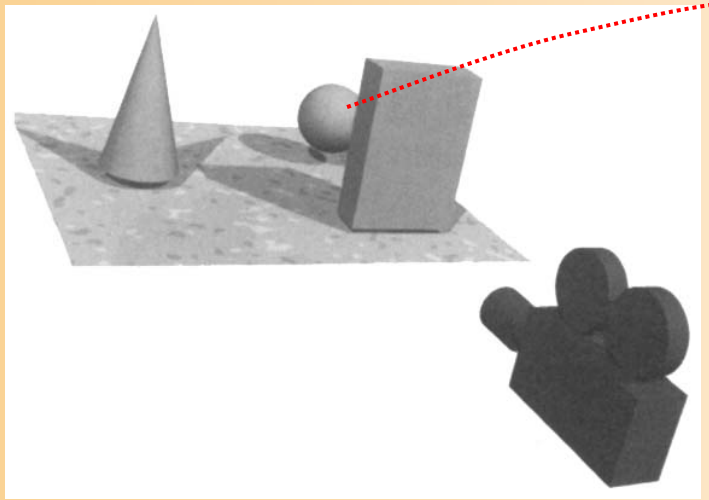
- レンダリング（Rendering）
  - カメラから見える画像を計算するための方法
  - 使用するレンダリングの方法によって、生成画像の品質、画像生成にかかる時間が決まる



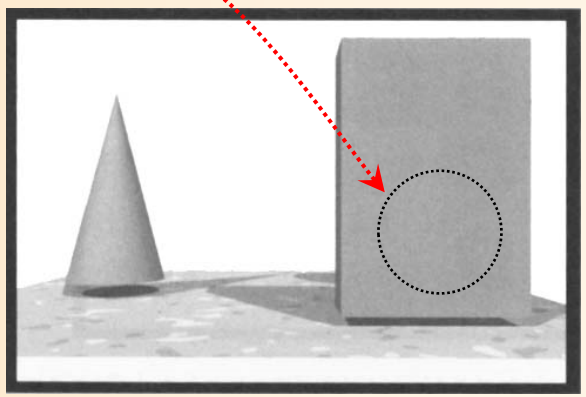


レンダリングの重要なポイント

- 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



この球は手前の直方体で隠れるため描画しない



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

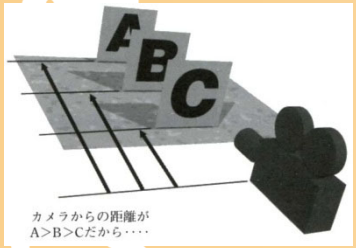
- 主なレンダリング手法
  - Zソート法
  - Zバッファ法
  - スキャンライン法
  - レイトレーシング法

低画質、高速度

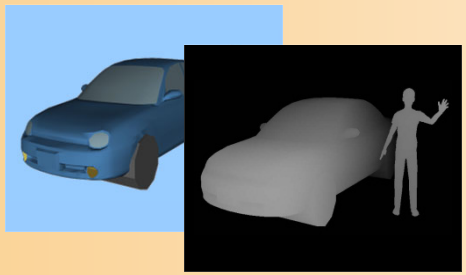
高画質、低速度

- 隠面消去の実現方法が異なる

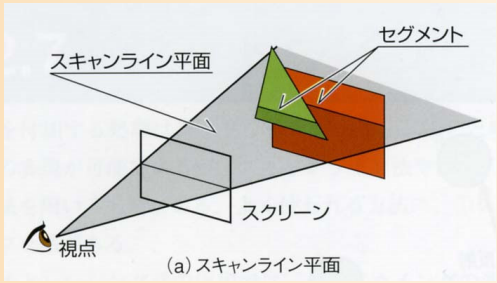
Zソート法



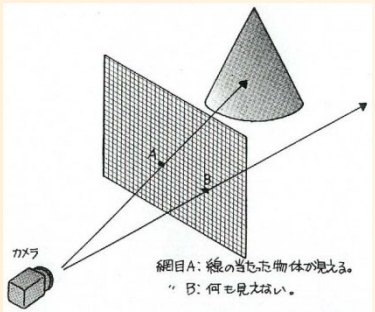
Zバッファ法



スキャンライン法

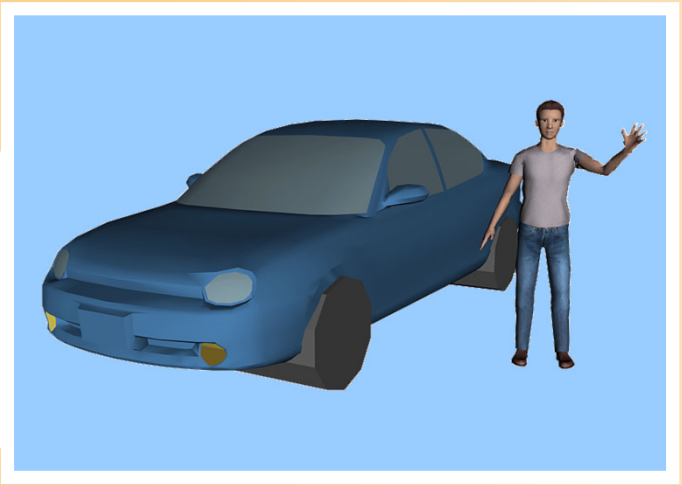


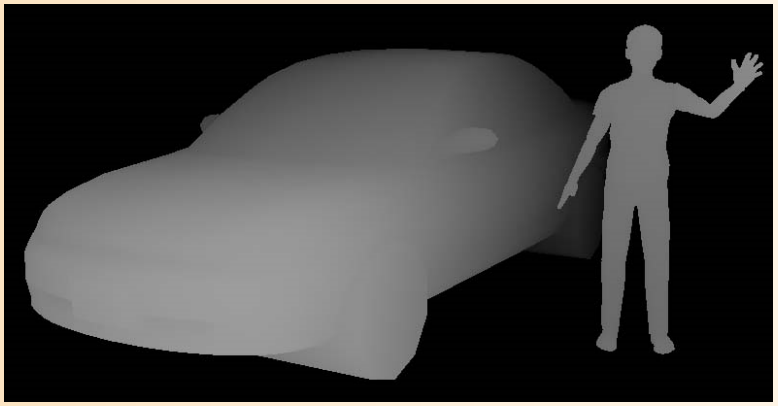
レイトレーシング法



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）

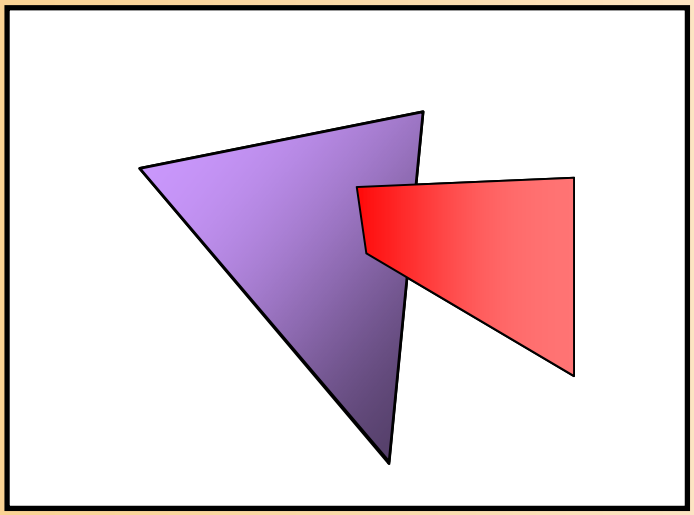


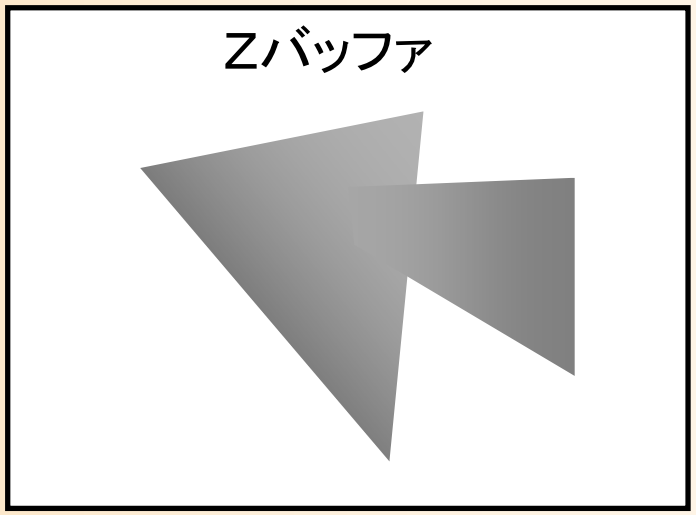


Zバッファの値（手前にあるほど明るく描画）

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

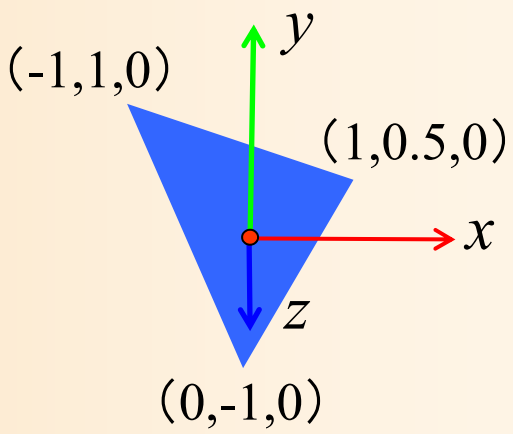




プログラムの例

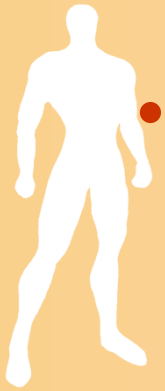
- １枚の三角形を描画するプログラムの例
  - 各頂点の頂点座標、法線、色を指定して描画

```
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

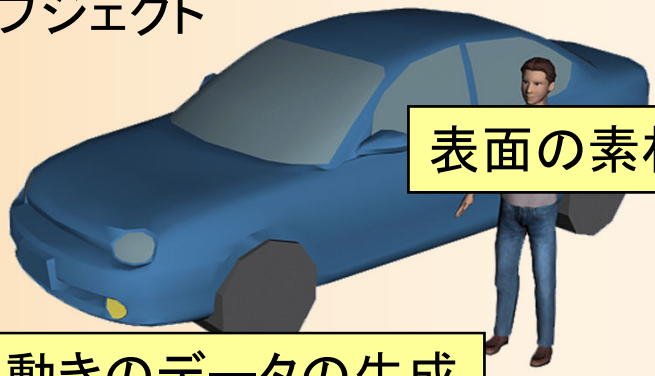
- モデリング
- レンダリング
- 座標変換
- シェーディング
- マッピング
- アニメーション



オブジェクトの作成方法

オブジェクトの形状表現

オブジェクト



表面の素材の表現

動きのデータの生成

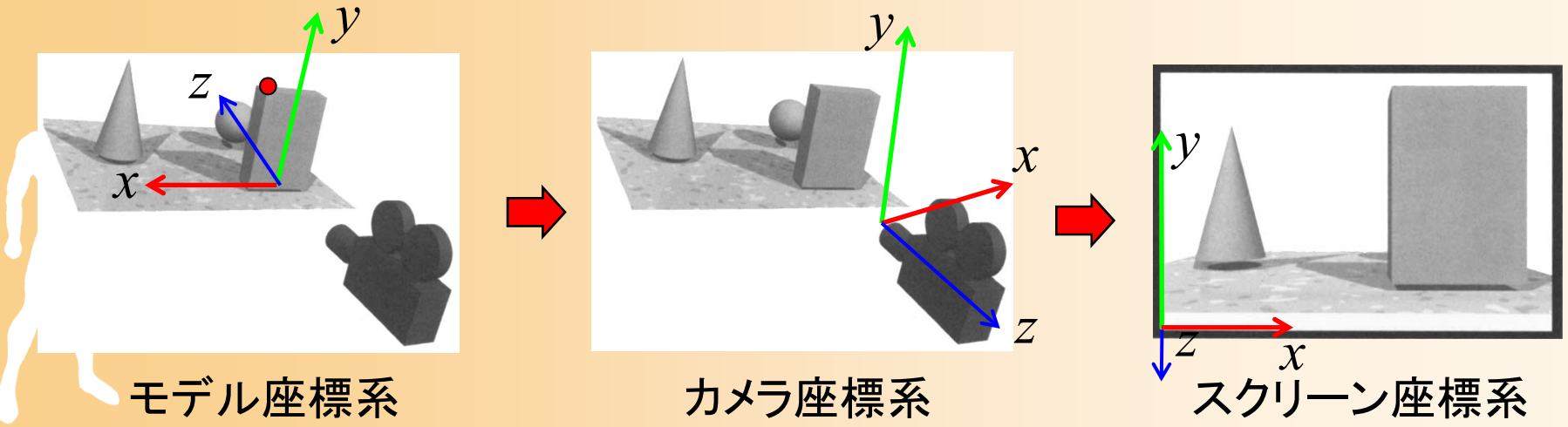
光源

カメラから見える画像を計算

光の効果の表現

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

アフィン変換

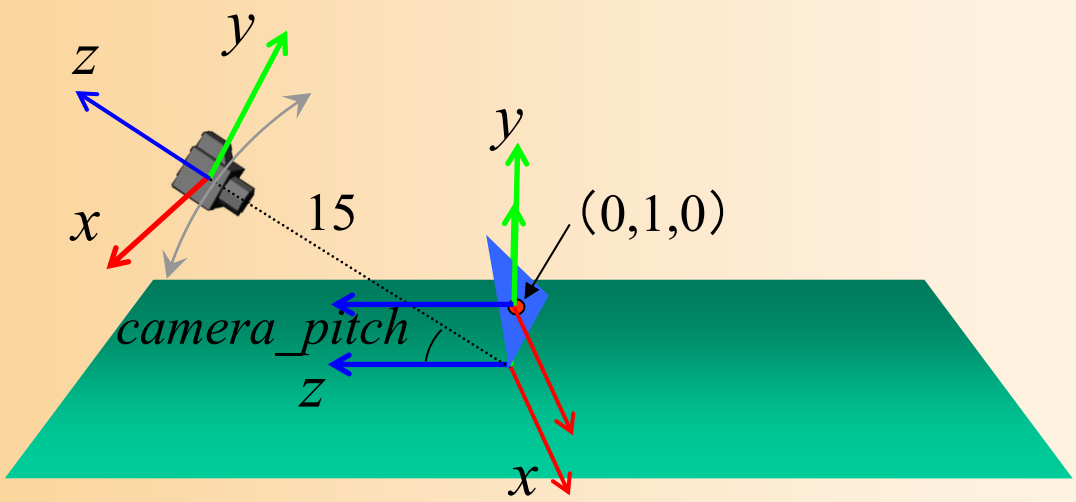
- アフィン変換（同次座標系変換）
  - ４×４行列の演算によって、３次元空間における回転・平行移動・拡大縮小などの処理を計算
  - 同次座標系
    - (x, y, z, w)の４次元座標値によって扱う
    - ３次元座標値は$(\mathrm{x/w}, \mathrm{y/w}, \mathrm{z/w})$で計算（通常は $\mathrm{w} = 1$）

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

  - 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

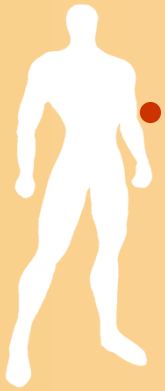
- 適切な変換行列を設定することで、カメラや物体の位置・向きを自在に変更できる

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

```
// 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



オブジェクトの作成方法

オブジェクトの形状表現

オブジェクト

表面の素材の表現

動きのデータの生成

光源

カメラから見える画像を計算

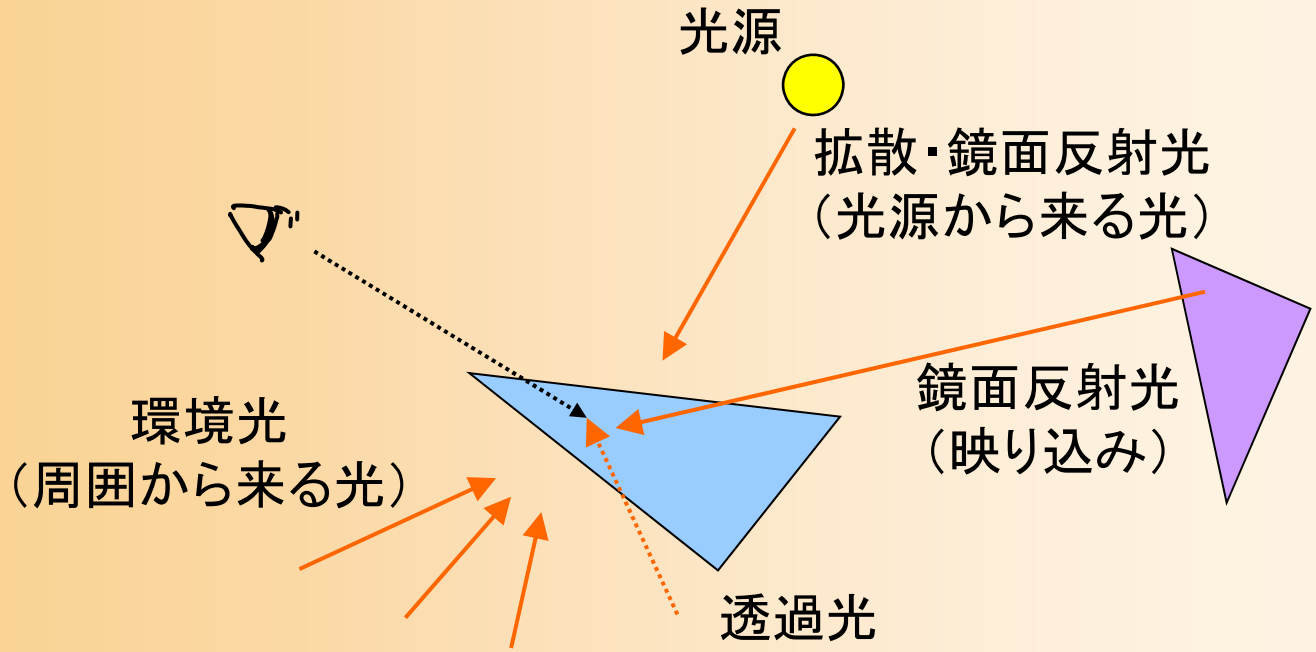
光の効果の表現

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

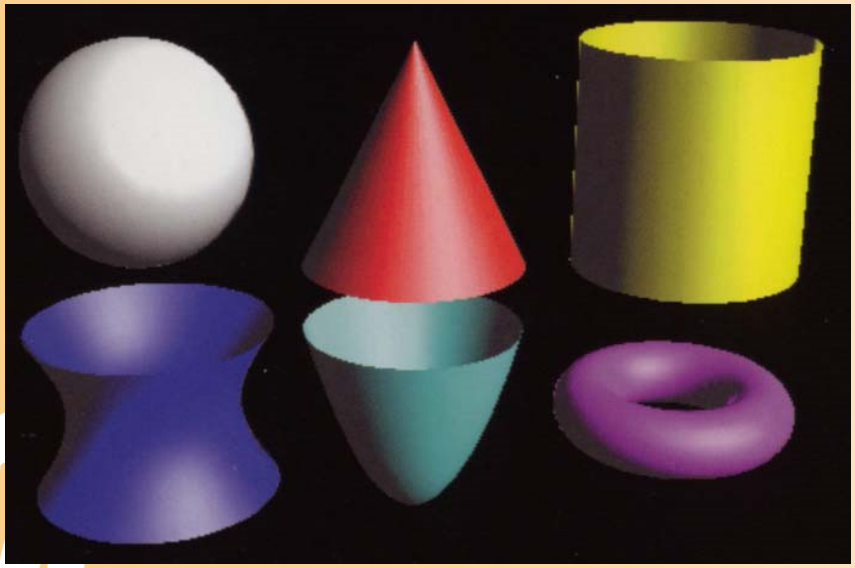
光のモデル

- 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



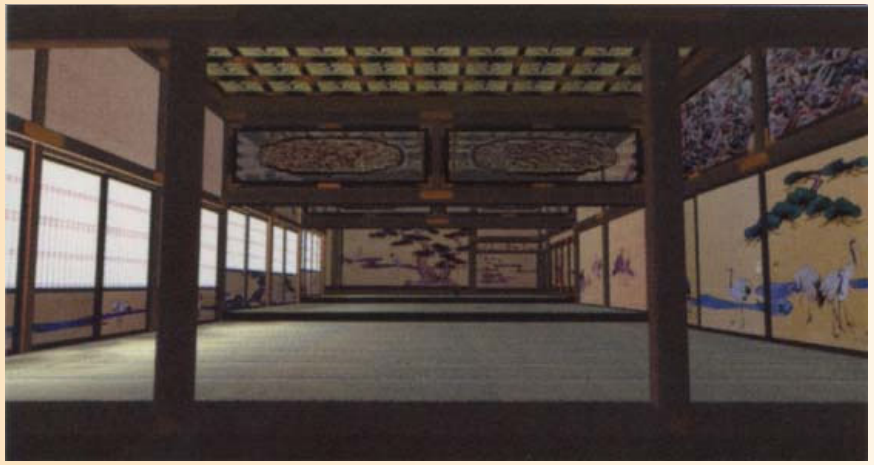
映り込み（大域照明）を考慮
基礎と応用 図8.9

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

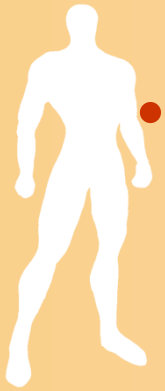
プログラムの例

- 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

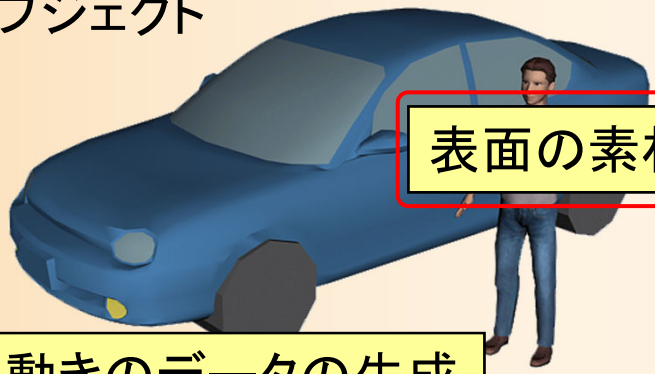
- モデリング
- レンダリング
- 座標変換
- シェーディング
- マッピング
- アニメーション



オブジェクトの作成方法

オブジェクトの形状表現

オブジェクト



表面の素材の表現

動きのデータの生成

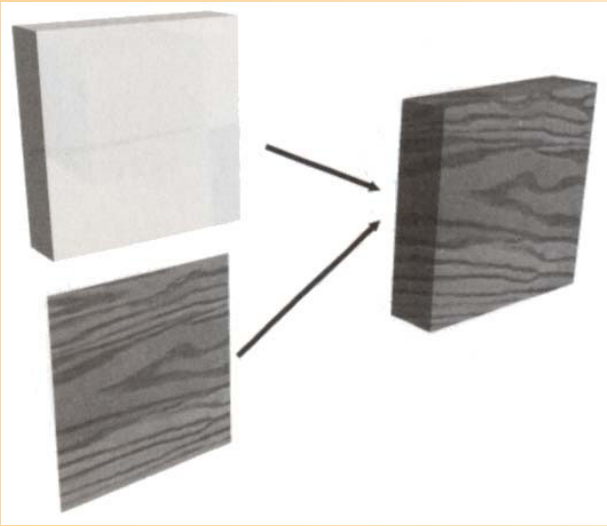
光源

カメラから見える画像を計算

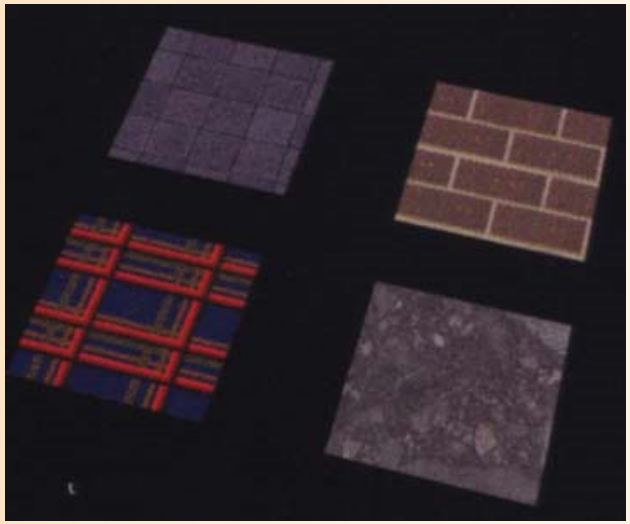
光の効果の表現

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



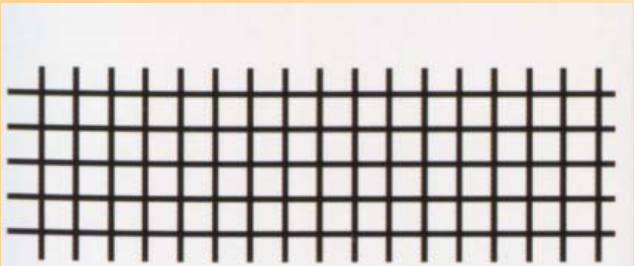
基礎知識 図3-19



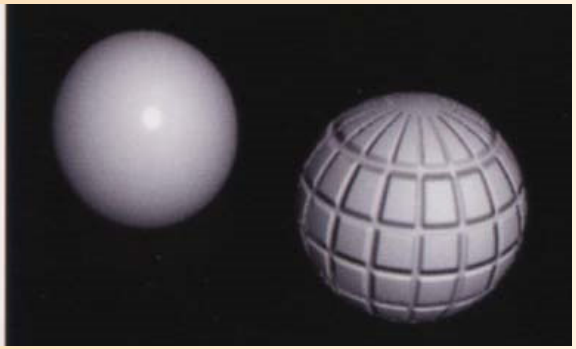
基礎と応用
図5.2

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

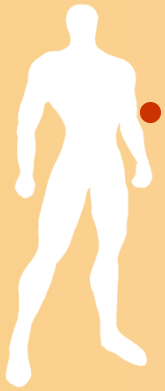


- 周囲の風景のマッピング（環境マッピング）

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



オブジェクトの作成方法

オブジェクトの形状表現

オブジェクト

表面の素材の表現

動きのデータの生成

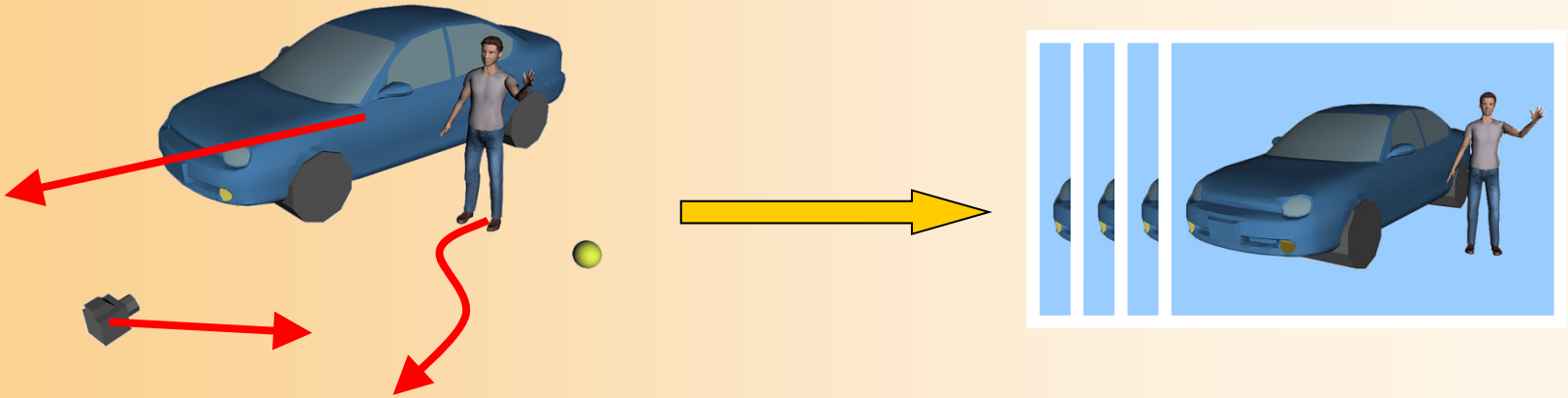
光源

カメラから見える画像を計算

光の効果の表現

アニメーション

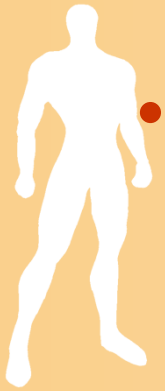
- 動きのデータをもとに、アニメーションを生成



- 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



オブジェクトの作成方法

オブジェクトの形状表現

オブジェクト

表面の素材の表現

動きのデータの生成

光源

カメラから見える画像を計算

光の効果の表現

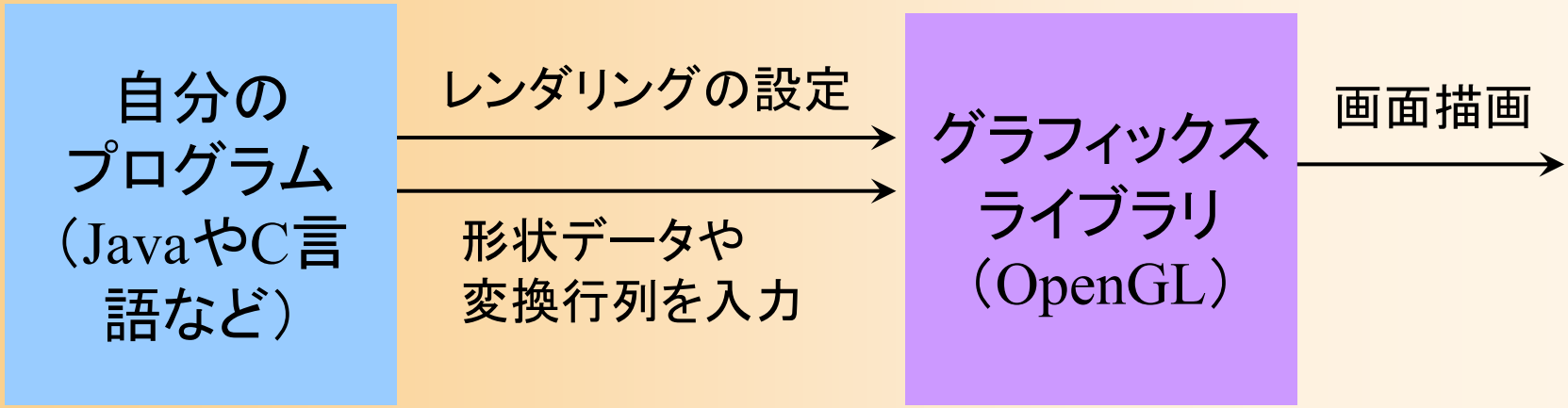
３次元グラフィックス・プログラミング

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

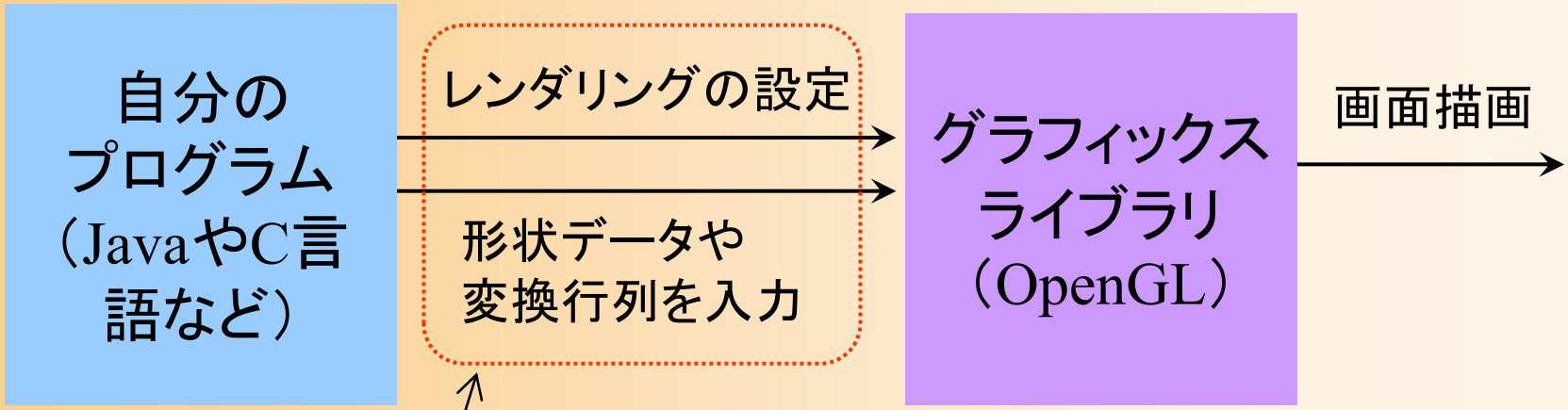
- 自分のプログラム と OpenGL の関係



レンダリング（＋座標変換、
シェーディング、マッピング）
などの処理を行ってくれる

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



最低限、これらの方法だけ学べば、
プログラムを作れる

これらの処理は、自分でプログラ
ムを作る必要はないが、しくみは
理解しておく必要がある

レンダリング（＋座標変換、
シェーディング、マッピング）
などの処理を行ってくれる

まとめ

- 前回の復習
- コンピュータグラフィックスの歴史と応用
- 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・プログラミング

次回予告

- ３次元グラフィックス・プログラミング演習（1）
  - OpenGLとGLUTの概要
  - サンプルプログラムの解説
  - プログラムのコンパイルと実行