コンピューターグラフィックスS

第11回　シェーディング

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
- レポート課題
- シェーディング
- 光のモデル
- スムーズシェーディング

今回の内容

- シェーディング
  - 光の効果の表現

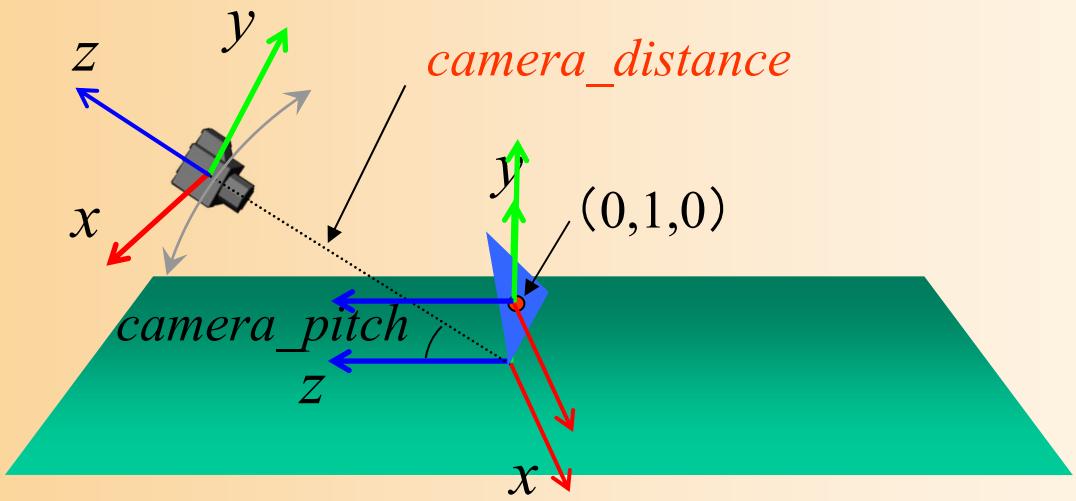
前回の復習

前回の演習

- 視点操作の拡張
- 変換行列によるアニメーション

視点操作の拡張

- 左ドラッグでカメラと注視点の距離を操作できるように拡張

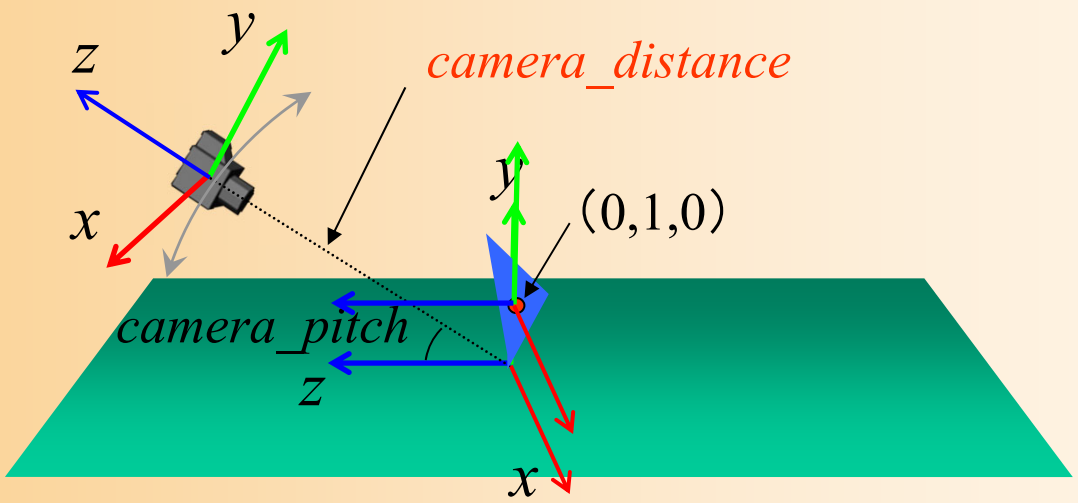


視点操作の拡張

- 変換行列

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

ワールド座標系→カメラ座標系　　モデル座標系→ワールド座標系



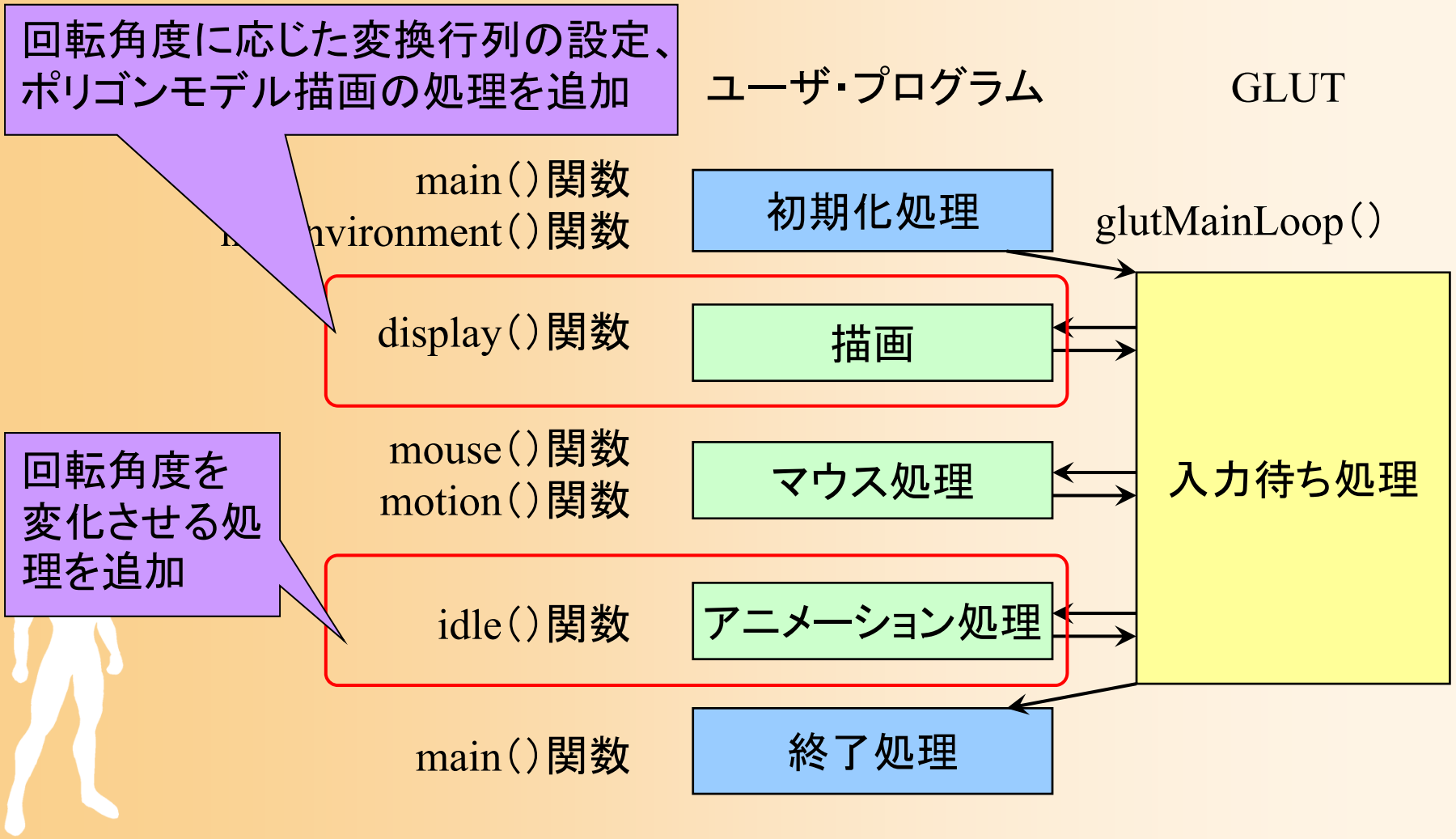
視点操作の拡張

- プログラムの修正箇所（多いので注意）
  - 左ボタンの押下状態を記録する変数を追加
  - カメラと原点の距離を記録する変数を追加
  - mouse()関数に、左ボタンの押下状態を更新する処理を追加
  - motion()関数に、左ドラッグに応じてcamera_distanceを変更する処理を追加
    - 一定値以上は近づかないように制限
  - display()関数を、camera_distanceに応じて変換行列を設定するように変更

変換行列によるアニメーション

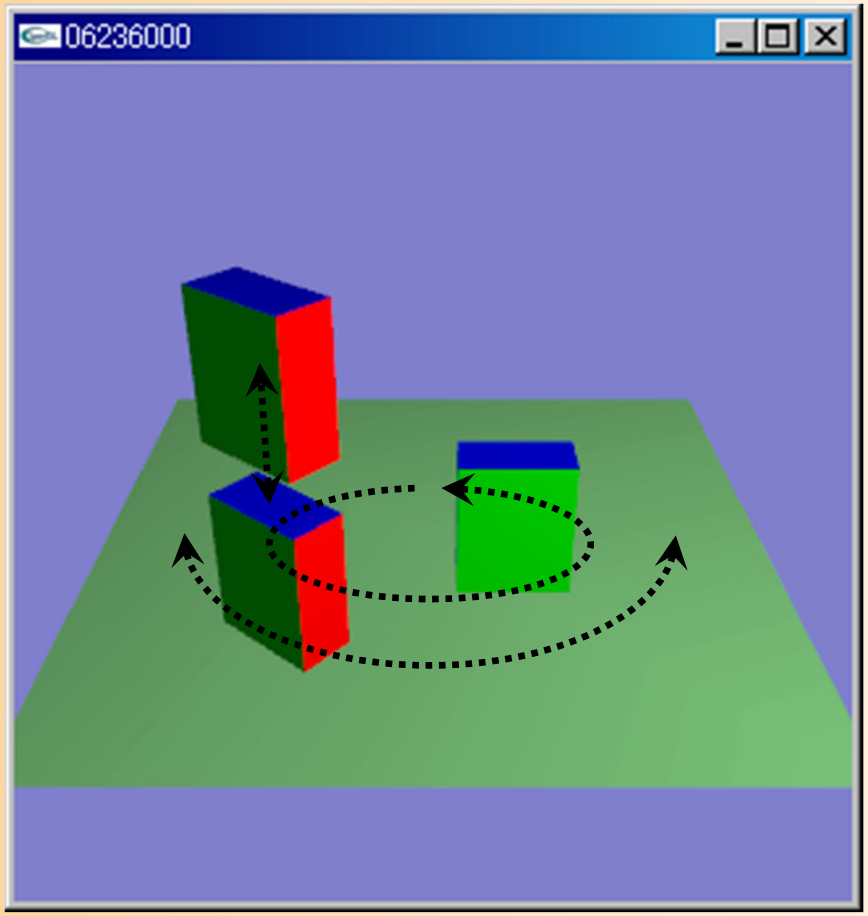
- 変換行列を組み合わせることで、さまざまな運動を実現できる
- idle()関数
  - 運動を表す媒介変数の変化を記述
- dysplay()関数
  - 媒介変数の値に応じて、回転角度や移動距離を設定

サンプルプログラムの構成



演習課題

- 演習課題のアニメーション



演習課題

```
void  display( void )
{
```

glPushMatrix() or glPopMatrix() or どちらも入れない

```
// 1つ目の直方体の描画（回転運動）
```

？

glPushMatrix() or glPopMatrix() or どちらも入れない

```
// 2つ目の直方体の描画（往復回転運動）
```

？

glPushMatrix() or glPopMatrix() or どちらも入れない

```
// 3つ目の直方体の描画（方物往復移動運動）
```

？

glPushMatrix() or glPopMatrix() or どちらも入れない

```
}
```

レポート課題

レポート課題

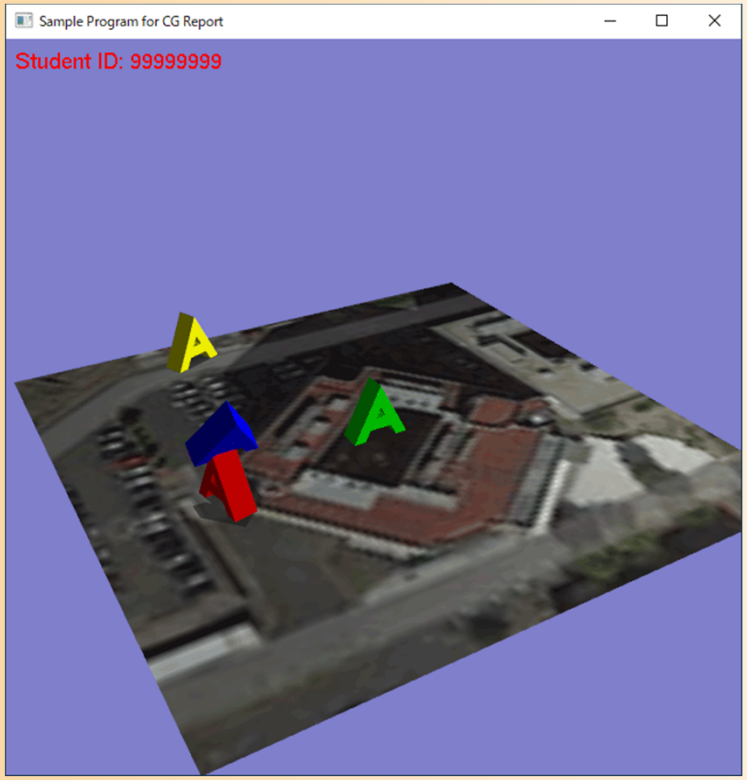
- OpenGLを使った課題プログラムの作成
  - 各自、自分に与えられた課題を実現するプログラムを作成する
    - ポリゴンモデルの描画、視点操作、アニメーション、テクスチャマッピング
- Moodleから提出
  - レポート、作成したプログラム一式を提出
- レポートの締め切りは後日連絡
  - 8月上旬（期末試験後）の締め切りを予定

レポート課題

- レポート課題
  1. ポリゴンモデルの描画
  2. 視点操作インターフェースの拡張
  3. アニメーション
  4. テクスチャマッピング
  - 各自、自分に与えられた課題を実現するプログラムを作成する
  - Moodleにアップロードされている課題のプログラムを実行
    - ログインIDにもとづいて学生番号・課題内容を表示

課題プログラム

- opengl_report2019.exe
  - 実行すると、各自の課題が表示される



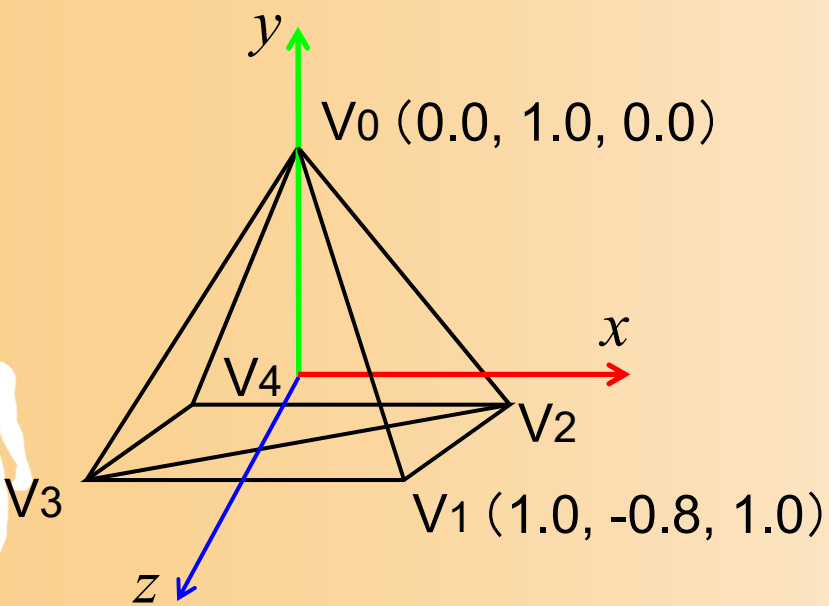

Sample Program for CG Report
Student ID: 99999999
OpenGLプログラミング レポート課題

1. ポリゴンモデルの描画

- 指定されたポリゴンモデルを描画する処理を追加すること
- レポートには、作成したポリゴンモデルが分かる情報を載せる
  - 各頂点の座標、面を構成する頂点番号の配列、各面の法線などの値
  - 頂点と面のつながりが分かる三面図や透視図
  - 単に、配列のデータを示すだけでは不十分
    - 何故、そのようなデータになったのか、結果だけでなく、導出の過程を説明する図が必要

ポリゴンモデルの情報の例(1)

- 四角すいを構成する頂点と三角面
  - 頂点座標
  - 三角面の頂点、面の法線



頂点座標

V0 (0.0, 1.0, 0.0)

V1 (1.0, -0.8, 1.0)

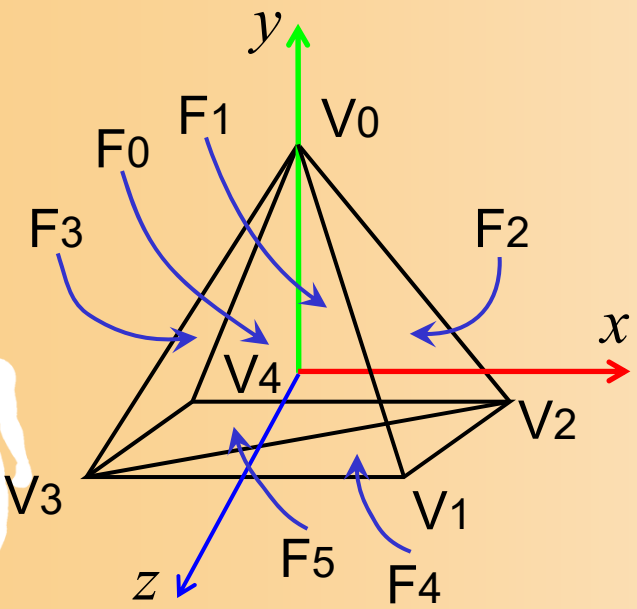
V2 (1.0, -0.8, -1.0)

V3 (-1.0, -0.8, 1.0)

V4 (-1.0, -0.8, -1.0)

ポリゴンモデルの情報の例(2)

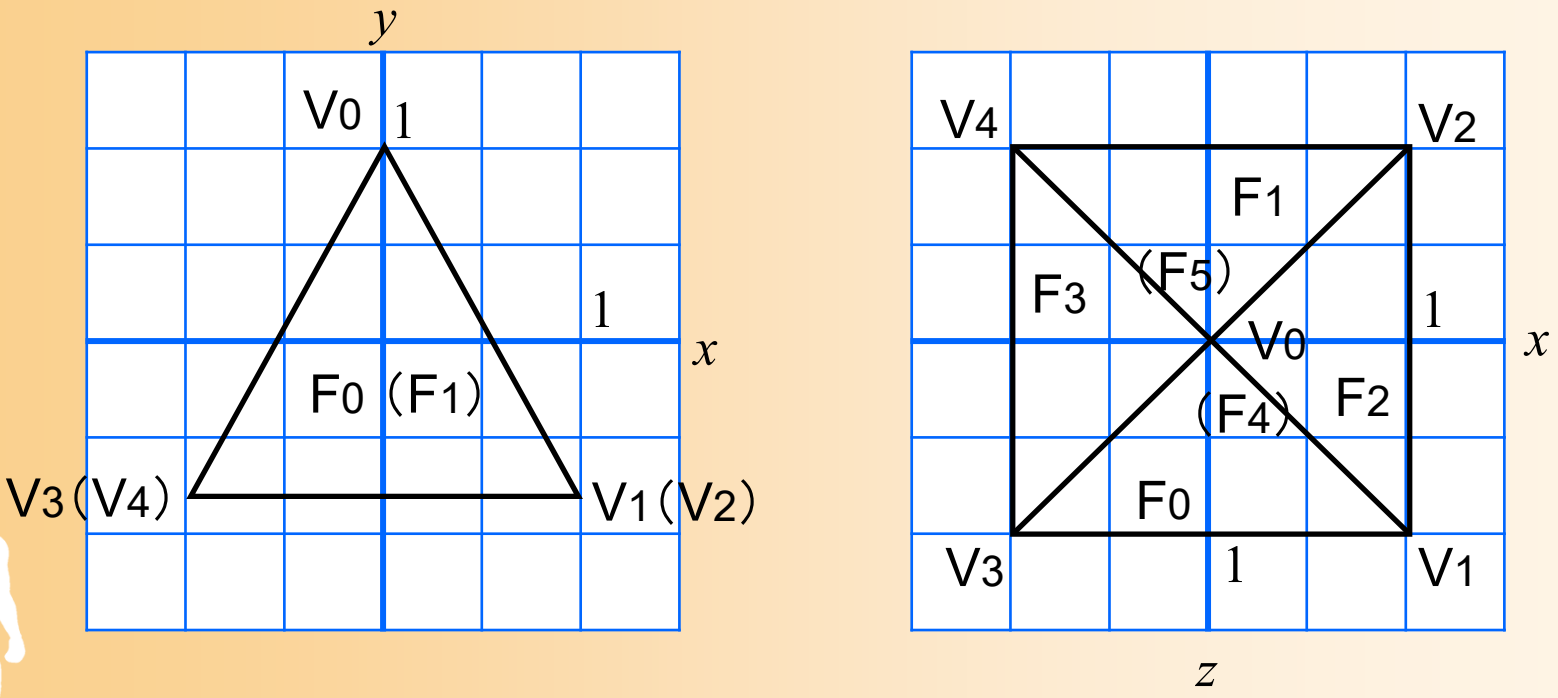
- 四角すいを構成する頂点と三角面
  - 頂点座標
  - 三角面の頂点、面の法線



三角面	法線
F0 { V0, V3, V1 }	{ 0.0, 0.53, 0.85 }
F1 { V0, V2, V4 }	{ 0.0, 0.53, -0.85 }
F2 { V0, V1, V2 }	{ 0.85, 0.53, 0.0 }
F3 { V0, V4, V3 }	{ -0.85, 0.53, 0.0 }
F4 { V1, V3, V2 }	{ 0.0, -1.0, 0.0 }
F5 { V4, V2, V3 }	{ 0.0, -1.0, 0.0 }

ポリゴンモデルの情報の例(3)

- 三面図(xy平面、xz平面)



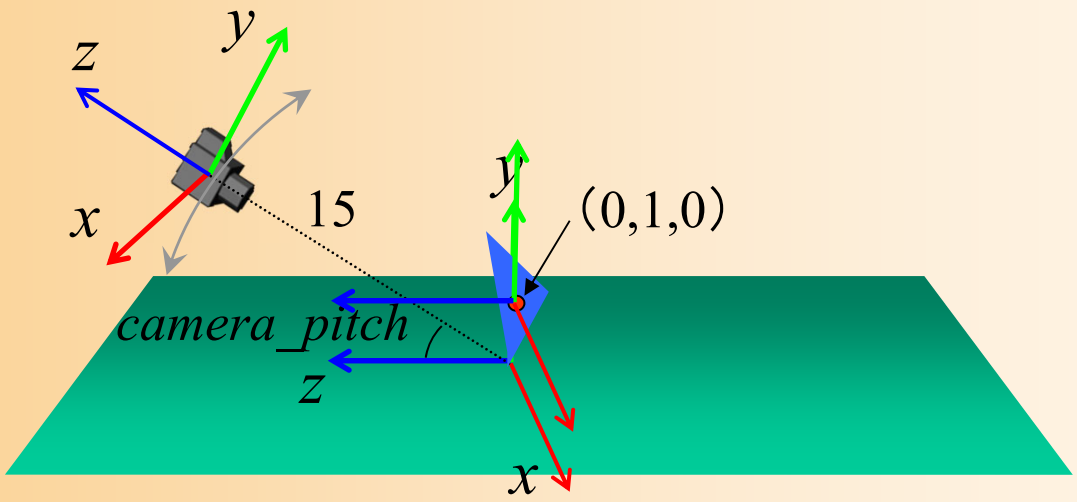
※ 括弧付きの頂点・面は裏側の頂点・面を表す

2. 視点操作インターフェース

- 視点操作インターフェースの機能を拡張するように、プログラムを変更する
- レポートには、視点操作のための変換行列を示すこと
  - 回転変換・平行移動変換の積の形で記述
  - 各変数の説明（どのような操作により、どのように変化する変数か）

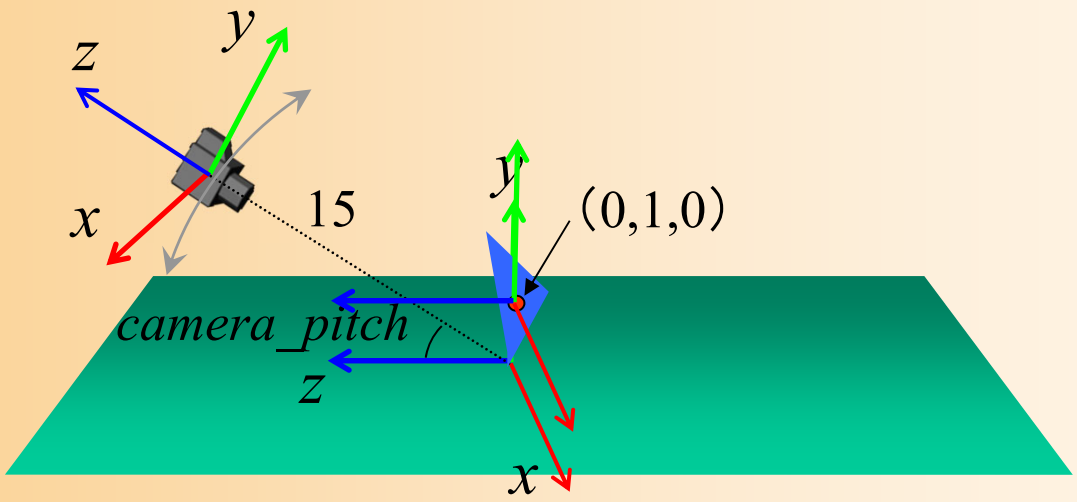
視点操作の変換行列の例(1)

- サンプルプログラムのシーン設定
  - カメラと水平面の角度(仰角)は camera_ptich
  - カメラと中心の間の距離は 15
  - ポリゴンを(0,1,0)の位置に描画



視点操作の変換行列の例(1)

- サンプルプログラムのシーン設定
  - カメラと水平面の角度(仰角)は camera_ptich
  - カメラと中心の間の距離は 15
  - ポリゴンを(0,1,0)の位置に描画



視点操作の変換行列の例(2)

- サンプルプログラムのシーン設定
  - カメラと水平面の角度(仰角)は camera_ptich
    - マウスの右ボタンを押しながら上下にドラッグすることで、−90〜0 の範囲内で変化

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

3. アニメーション

- 指定された通りの、複数の物体のアニメーションを実現する
- レポートには、課題2と同様、どのような変換行列を使用したかを示すこと

オブジェクトの変換行列の例

- 各物体がどのような運動を行うかの説明
  - 使用する変数とそれぞれの変化を説明

物体1

$$\left(\mathbf{M} \right) \begin{pmatrix} \cos(theta\_cycle) & 0 & \sin(theta\_cycle) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(theta\_cycle) & 0 & \cos(theta\_cycle) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1.5 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}$$

物体2

$$\left(\mathbf{M} \right) \begin{pmatrix} \cos(theta\_cycle2) & 0 & \sin(theta\_cycle2) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(theta\_cycle2) & 0 & \cos(theta\_cycle2) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}$$

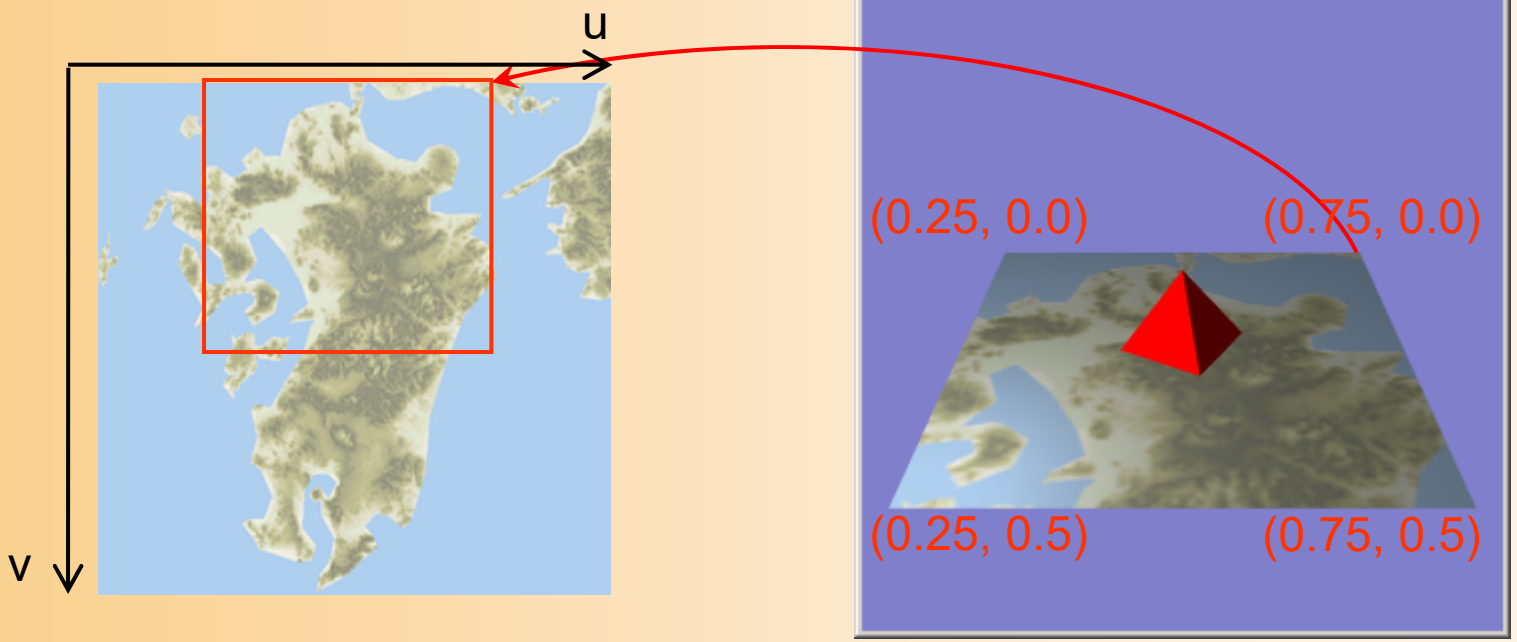
ワールド座標系→カメラ座標系（共通）

4. テクスチャマッピング

- 指定された通りに、あらかじめ用意されている画像を読み込み、その一部を、地面にマッピングする
- レポートには、テクスチャ座標の範囲や計算方法を示すこと

テクスチャ座標の計算の例(1)

- テクスチャマッピングの範囲の説明
  - テクスチャ画像のどの範囲をポリゴンに貼り付けるか

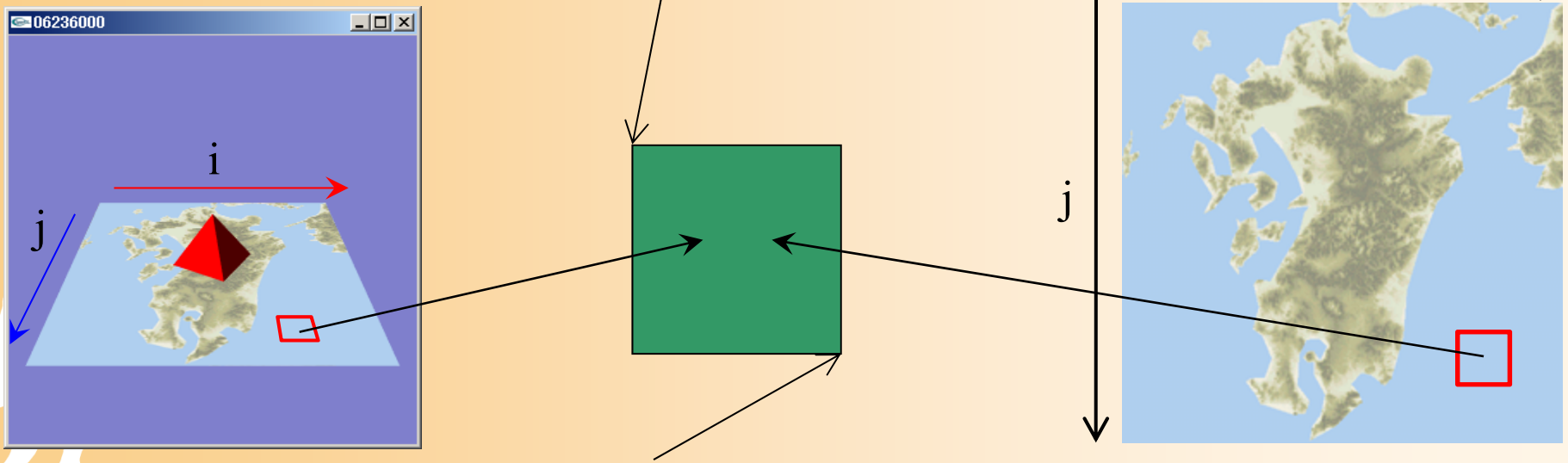


テクスチャ座標の計算の例(2)

- 10×10の四角形を並べて描画
  - 頂点位置とテクスチャ座標を i, j から計算

頂点位置（i＊1 - 5, 0, j＊1 - 5）

テクスチャ座標（i＊0.1 – 5, j＊0.1 - 5）



頂点位置（(i+1)＊1 – 5, 0, (j+1)＊1 – 5）

テクスチャ座標（(i+1)＊0.1 + 0.1, (j+1) * 0.1）

レポート課題に関する注意（1）

- くれぐれも十分早くから準備を始めること
  - 締め切りの直前に始めても、間に合わない
  - 学期末には、他の科目のレポートも重なるので、まとめてやろうとしても間に合わない
  - きちんと計画的に課題に取り組むこと
  - 少なくとも締め切りの１週間前までには課題の内容は終らせて、残りの時間はレポート作成に使った方が良い
  - 何か質問や相談等があれば、早めに申し出ること（締め切り直前になって来ても間に合わない）

レポート課題に関する注意(2)

- レポートをきちんと書くこと
  - どのようにして処理を実現しているのかが、きちんと分かるように書く(きちんと文章で説明できることも重要)
  - 見出しや段落分け、適切な余白・行間、プログラムの引用は枠で囲むなど、レポートの見やすさも重要
- プログラムも見やすく書くこと
  - 演習で作成したプログラムのうち、レポート課題に不要なものは、削除すること
  - コメントやインデントを適切につけること

レポート課題に関する注意(3)

- 不正行為は絶対にしないこと
  - たとえ一部でも、他人のプログラム・レポートを丸写しした場合は、不正行為となり、厳重に処罰・減点される

よくある間違い(1)

- 指定されたプログラムが作成されていない
  - ポリゴンモデル
    - 法線が間違っている(光源から照らされた面の色がおかしい)
    - 隣接するポリゴンが辺を共有していない、など
  - 視点操作
  - アニメーション
    - 動きの周期(物体同士の回転周期)が異なる、など
  - プログラムがコンパイルできない(論外)
    - 提出直前の修正や、ファイルの間違いなどに注意
    - 間違ったプログラムの書き方

よくある間違い(2)

- レポートの書き方
  - 説明不足のレポートが多い
    - ポリゴンモデルの図、変換行列の導出理由、各処理の実現方法
    - 作成結果だけではなく、何故そのようなプログラムを作成したのか、理由の説明が必要
  - レポートの見易さ・読みやすさ、レポートの体裁
    - 適切な余白・行間、章分け・段落分け、インデント、本文と引用プログラムの区別、誤字脱字、など
- 詳細は、レポート課題の説明も参照すること

演習環境

- 夏休み期間中（試験期間終了後）は、マルチメディア講義室は閉室され、利用できないので、注意する
- CL端末室（Windows環境）や情報科学センタの端末室（Linux環境）で演習を行うこともできる
  - プログラミング環境が異なるので、注意する

シェーディング

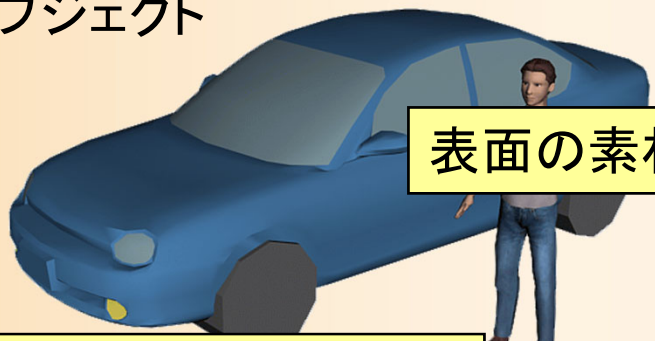
今回の内容

- シェーディング
  - 光のモデル
  - スムーズシェーディング

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

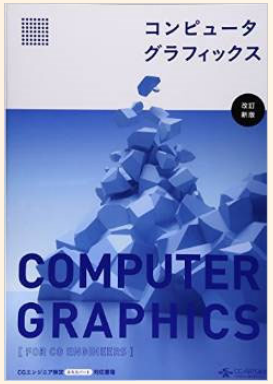
光源

光の効果の表現

カメラから見える画像を計算

画像処理

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
  - 4章 4-3～4-4
- 「ビジュアル情報処理 ―CG・画像処理入門―」
  CG-ARTS協会 編集・出版
  - 4章 4-3～4-4

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）
  - 4章
- 「3DCGアニメーション」
  栗原恒弥 安生健一 著、技術評論社 出版
  - 第2章（68～108ページ）
- 「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
  - 第2章（23～28ページ）、第4章（35～39ページ）、
    第5章（40～49ページ）、第8章（73～75ページ）、
    第9章（79～90ページ）

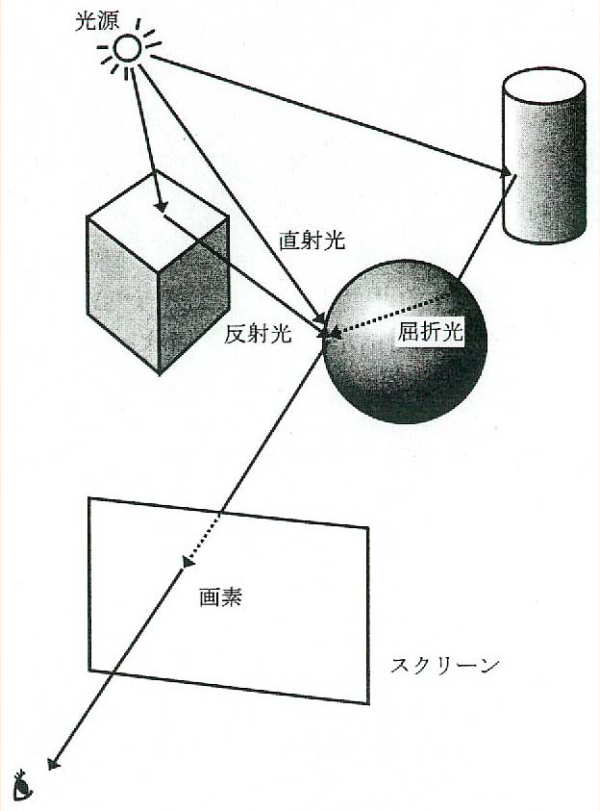
シェーディング

- 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

光のモデル

- 光のモデル
  - 物体に光が当たることによって物体の色(輝度)が決まる
- 光を種類に分けて考える
  - 環境光
  - 反射光
    - 拡散反射光
    - 鏡面反射光
  - 透過光



基礎と応用 図8.2

光のモデル

- 輝度の計算式
  - 全ての光による影響を足し合わせることで、物体上の点の輝度（RGBの値）が求まる
    - 各Iは光の明るさ（RGB）
    - 各kは物体の反射特性（RGB）

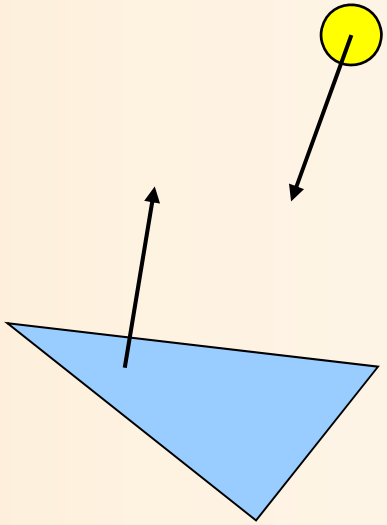
$$I = I_a k_a + \sum_{i=1}^{n_L} I_i \left[k_d \left(N \cdot L \right) + k_s \left(R \cdot V \right)^n \right] + k_r I_r + k_t I_t$$

環境光　拡散反射光　鏡面反射光（局所照明）　鏡面反射光（大域照明）　透過光

それぞれの光源からの光（局所照明）　大域照明

光のモデル

- 局所照明モデル
  - 光源と一枚の面の関係のみを考慮したモデル
    - 環境光、拡散反射光、鏡面反射光



- 大域照明モデル
  - 周囲の物体の影響も考慮したモデル
    - 環境光、鏡面反射光、透過光

  - 同じ種類の光でも考慮する範囲に応じて局所モデルと大域モデルがあるので注意

光のモデルの要素

- 環境光（局所照明・大域照明モデル）
- 拡散反射光（局所照明モデル）
- 鏡面反射光（局所照明モデル）
- 鏡面反射光（大域照明モデル）
- 透過光（大域照明モデル）

$$I = I_a k_a + \sum_{i=1}^{n_L} I_i \left[k_d \left(N \cdot L \right) + k_s \left(R \cdot V \right)^n \right] + k_r I_r + k_t I_t$$

環境光　拡散反射光　鏡面反射光（局所照明）　鏡面反射光（大域照明）　透過光

それぞれの光源からの光（局所照明）　大域照明

光のモデルの要素

- 環境光（局所照明・大域照明モデル）
- 拡散反射光（局所照明モデル）
- 鏡面反射光（局所照明モデル）
- 鏡面反射光（大域照明モデル）
- 透過光（大域照明モデル）

環境光

- 環境光（局所照明モデル）
  - 環境全体の光（周囲からくる光）
    - 光源からの光が周囲の面に拡散反射した光
  - きちんと計算しようとすると非常に複雑になってしまうので、基本的には明るさ一定として処理
    - 基本的には、屋外や室内といった環境により決まる
    - 実際には、同じ室内でも、部屋の窓際は少し明るく、廊下側は暗いといった、明るさの違いが出る
- 環境光（大域照明モデル）
  - 環境光をより正しく計算するための手法もある（ラジオシティ法、フォトンマップ法）

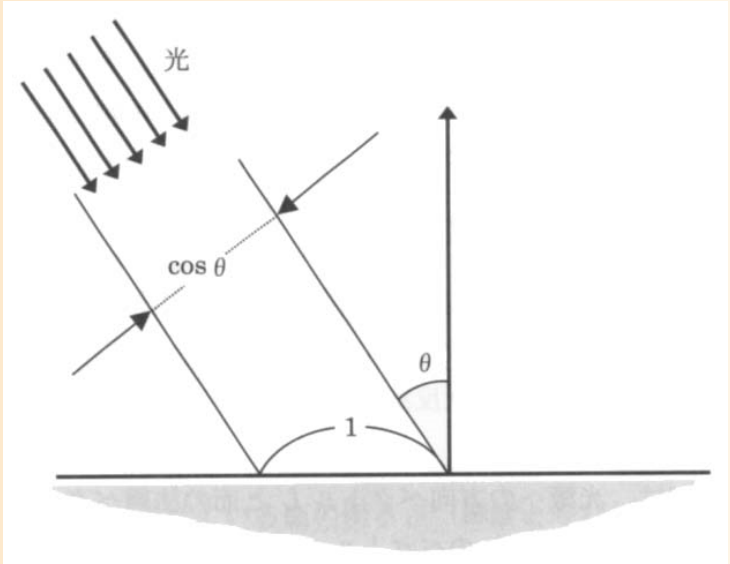
光のモデルの要素

- 環境光（局所照明・大域照明モデル）
- 拡散反射光（局所照明モデル）
- 鏡面反射光（局所照明モデル）
- 鏡面反射光（大域照明モデル）
- 透過光（大域照明モデル）

反射光

- 拡散反射光（局所照明モデル）
  - ランバート（Lambert）の余弦則
  - 光源と一枚の面の位置関係だけを考慮
  - 光の方向と面の法線の間の角度によって決まる
    - 両者の向きが近いほど明るくなる

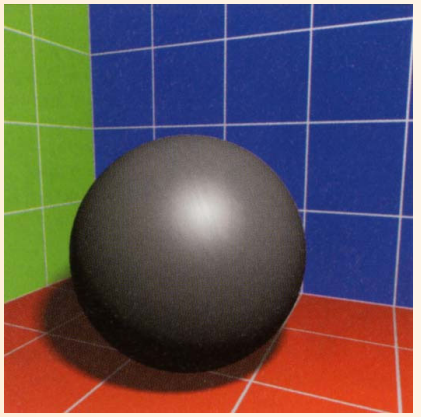
$$I_d = k_d I_l \cos\theta$$
$$= k_d I_l (N \cdot L)$$



基礎と応用 図2.9

反射光

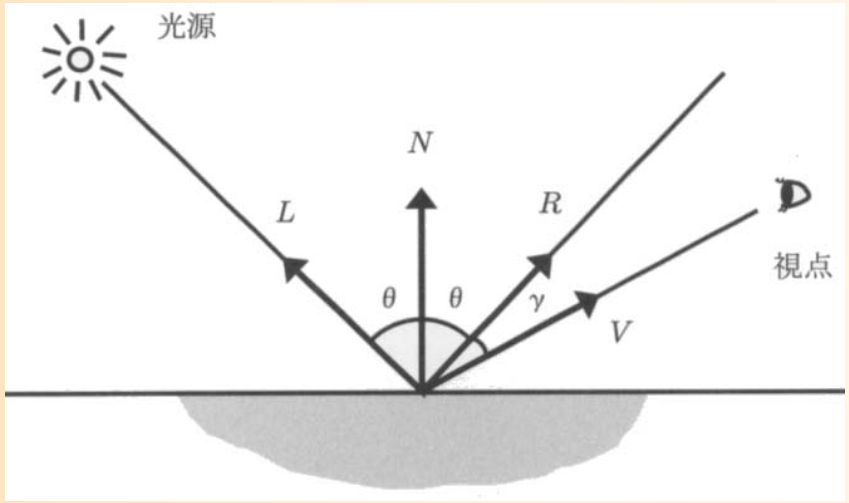
ハイライトの例



- 鏡面反射光（局所照明モデル）
  - フォン（Phong）のモデル
  - ハイライトの表現
  - 光の入射角と視線の角度が一致する時に最も明るくなる

$$I_d = k_s I_l \cos^n \gamma$$

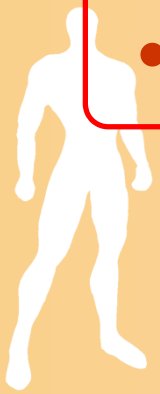
$$= k_s I_l \left(R \cdot V \right)^n$$



基礎と応用 図2.10

光のモデルの要素

- 環境光（局所照明・大域照明モデル）
- 拡散反射光（局所照明モデル）
- 鏡面反射光（局所照明モデル）
- 鏡面反射光（大域照明モデル）
- 透過光（大域照明モデル）



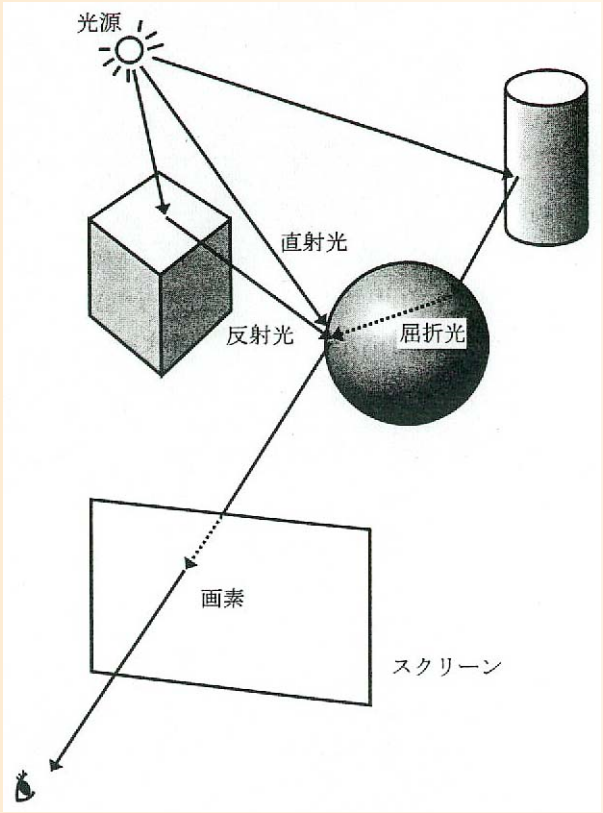
反射光

- 鏡面反射光(大域照明モデル)
  - レイトレーシングによるレンダリングの時に考慮
  - 反射先の物体上の点の色をI_rとする

$$I = k_r I_r + k_t I_t$$

鏡面反射光 透過光
(大域照明)

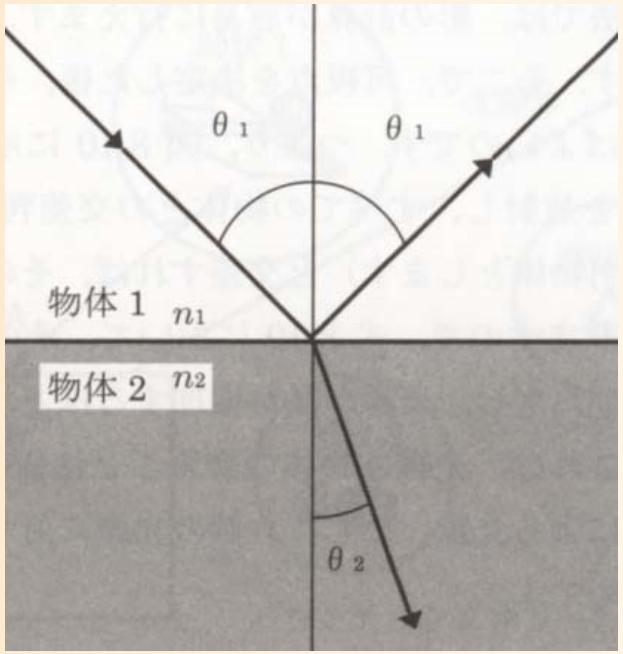
大域照明



基礎と応用 図8.2

透過光

- 透過光（大域照明モデル）
  - レイトレーシングによるレンダリングの時に考慮
  - 透明な物体を通り抜けてくる光
  - 屈折の影響も考慮
    （スネル（Snell）の法則）



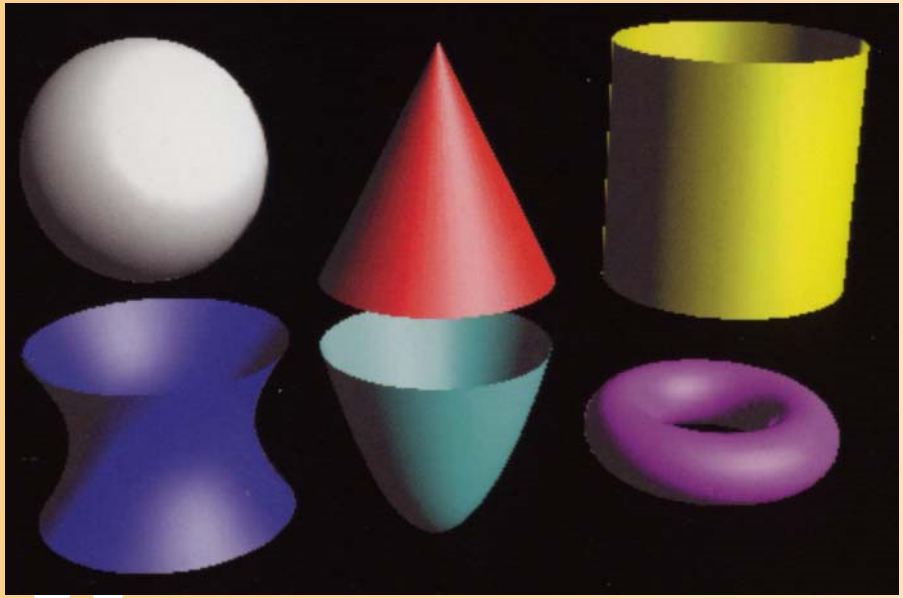
基礎と応用 図8.8

$$n_1 \sin\theta_1 = n_2 \sin\theta_2$$

（n_1, n_2 は物体の屈折率）

大域照明モデルの例

- 局所照明のみの場合と、大域照明（反射）を含む場合の比較



局所照明のみ
基礎と応用 図1.1

大域照明を考慮
基礎と応用 図8.9

光のモデルのまとめ

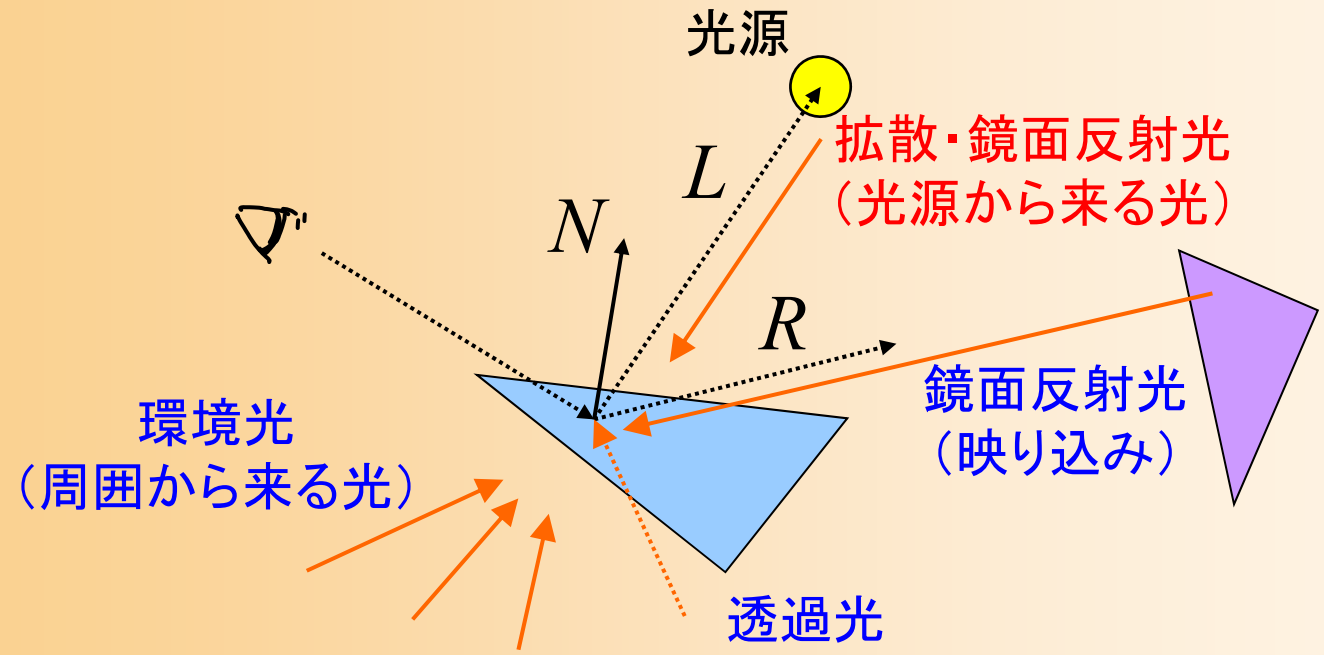
- 輝度の計算式
  - 全ての光による影響を足し合わせることで、物体上の点の輝度（RGBの値）が求まる
    - 各Iは光の明るさ（RGB）
    - 各kは物体の反射特性（RGB）

$$I = I_a k_a + \sum_{i=1}^{n_L} I_i \left[k_d \left(N \cdot L \right) + k_s \left(R \cdot V \right)^n \right] + k_r I_r + k_t I_t$$

環境光　拡散反射光　鏡面反射光（局所照明）　鏡面反射光（大域照明）　透過光

それぞれの光源からの光（局所照明）　大域照明

光のモデルのまとめ



$$I = I_a k_a + \sum_{i=1}^{n_L} I_i \left[k_d \left(N \cdot L \right) + k_s \left(R \cdot V \right)^n \right] + k_r I_r + k_t I_t$$

環境光　拡散反射光　鏡面反射光（局所照明）　鏡面反射光　透過光（大域照明）

それぞれの光源からの光（局所照明）　大域照明

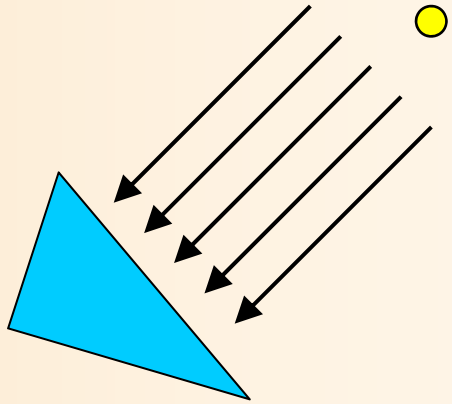
光源の種類

- 平行光源
- 点光源
- スポットライト光源
- 線光源、面光源
  - 局所照明モデルのために、いくつかの単純な光源モデルを使用
  - 下のものほど複雑で計算量が大きくなる
  - 表現したい光源に応じて適切なモデルを使用

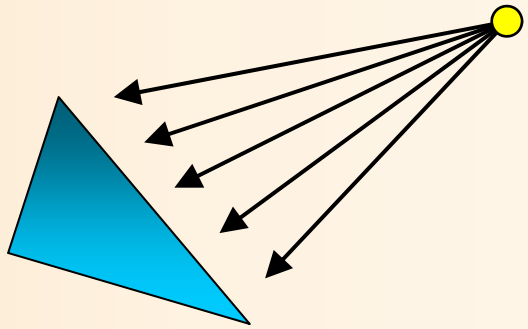
光源の種類

- 平行光源
  - 一定方向からの光源
  - 計算量が最も少ない
  - 太陽などの遠くにある光源の表現に適している

無限遠に光源があると見なす



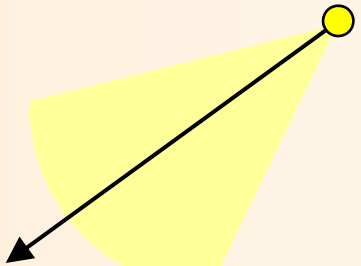
- 点光源
  - 位置の決まった光源
  - ライトなどの表現に適している
  - 光の方向は点光源と面の位置関係により決まる
  - 光の減衰も考慮できる

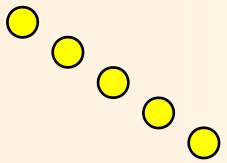


光源の種類

- スポットライト光源
  - 位置と向きの決まった光源
  - スポットライトなどの表現に適している
- 線光源、面光源
  - 沢山の点光源の集まりと考える
  - 非常に計算時間がかかる
  - 蛍光灯などの表現に使用

指定した方向・角度にのみ有効な点光源





光のモデルの計算

- OpenGLによる光の効果の計算
  - 設定された光源情報、及び、法線に従い、局所照明モデルのみを計算
    - 環境光には、一定の明るさを指定可能
    - 光源の種類や位置を指定可能（複数指定可能）

$$I = I_a k_a + \sum_{i=1}^{n_L} I_i \left[k_d \left(N \cdot L \right) + k_s \left(R \cdot V \right)^n \right] + k_r I_r + k_t I_t$$

環境光　拡散反射光　鏡面反射光（局所照明）　鏡面反射光（大域照明）　透過光

それぞれの光源からの光（局所照明）　大域照明

プログラムの例

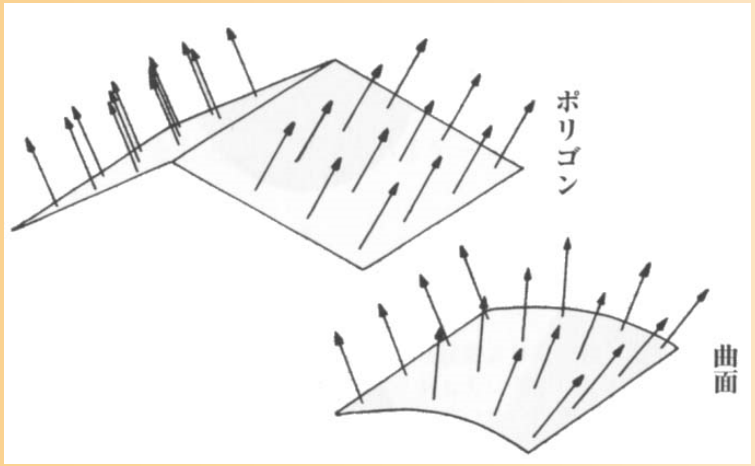
- 光源の位置や色の設定（詳細は後日の演習）
  - 以下の例では、環境光と、一つの点光源を設定

```
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 );
```

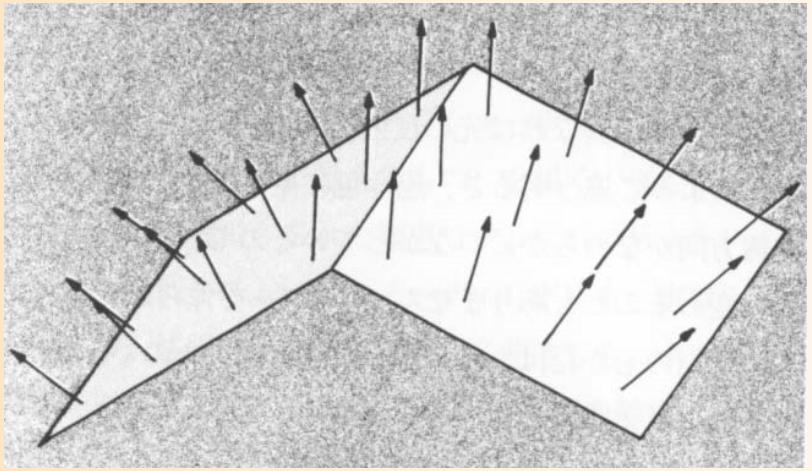
スムーズシェーディング

スムーズシェーディング

- ポリゴンモデルに当たる反射光（局所照明モデル）の効果をきれいに表現する技術
  - ポリゴンモデルを単純に描画すると、面ごとに色が急に変わることになり、角ばって見えてしまう
  - 面の向き（法線）を操作して、なめらかに見せる



基礎知識 図3-12



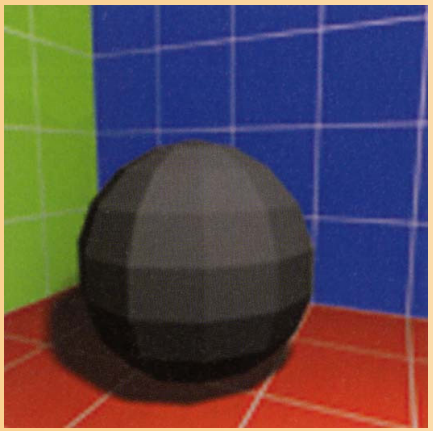
基礎知識 図3-13

シェーディングの方法

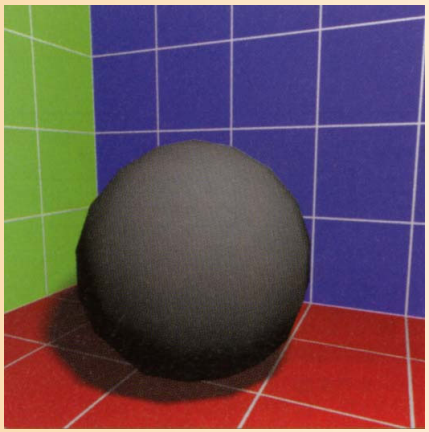
- フラットシェーディング
  - 単純に面の法線によって面の色をきめる方法
- スムーズシェーディング
  - 面の色をなめらかに変化させる方法
  - 2種類の方法がある
    - グローシェーディング
    - フォンシェーディング

シェーディングの方法

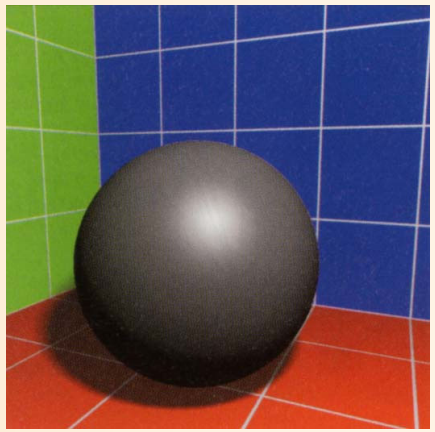
- フラットシェーディング
- スムーズシェーディング
  - グローシェーディング
  - フォンシェーディング



フラットシェーディング



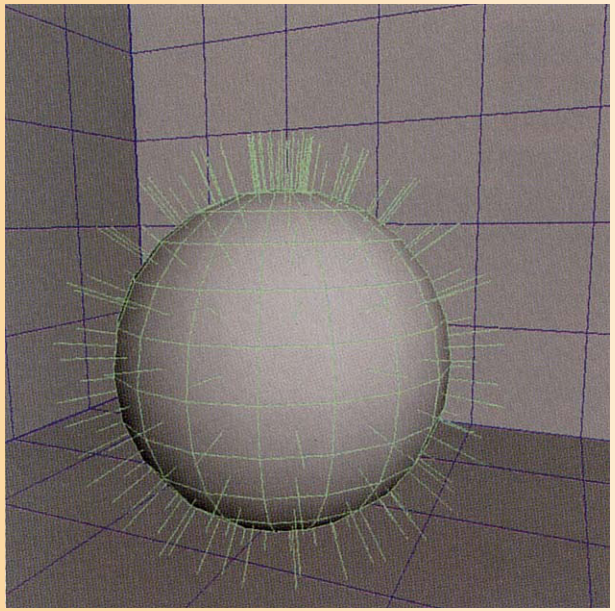
グローシェーディング

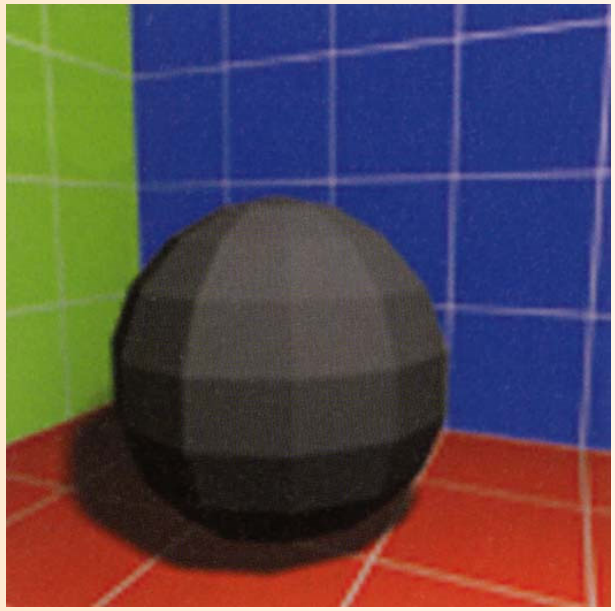


フォンシェーディング

フラットシェーディング

- フラットシェーディング（Flat Shading）
  - 面の法線をもとに面の色を決定
  - 単一色で面を描画
  - 面と面の境界がはっきりと分かってしまう





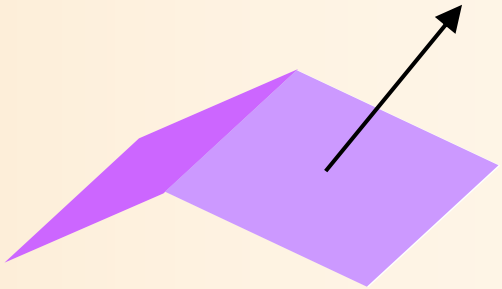
CG制作独習事典
p.4 p.16

フラットシェーディング

- フラットシェーディングの手順
  - 面の法線をもとに面の色を決定
    - 先ほど説明した、光源処理を適用

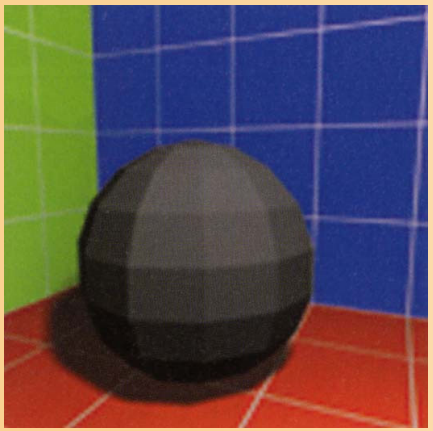
$$I = I_a k_a + \sum_{i=1}^{n_L} I_i \left[k_d \left(N \cdot L \right) + k_s \left(R \cdot V \right)^n \right] + k_r I_r + k_t I_t$$

  - 面の色に従って、面内の各ピクセルを描画

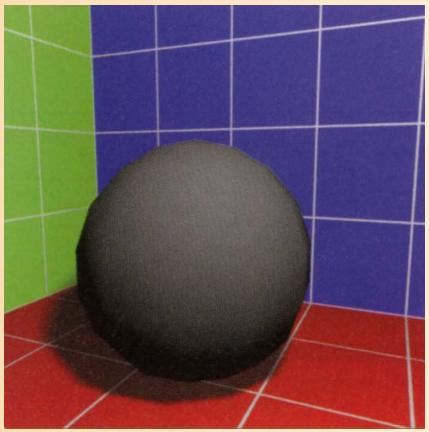


シェーディングの方法

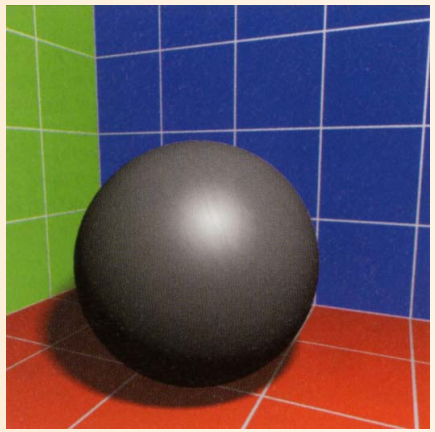
- フラットシェーディング
- スムーズシェーディング
  - グローシェーディング
  - フォンシェーディング



フラットシェーディング



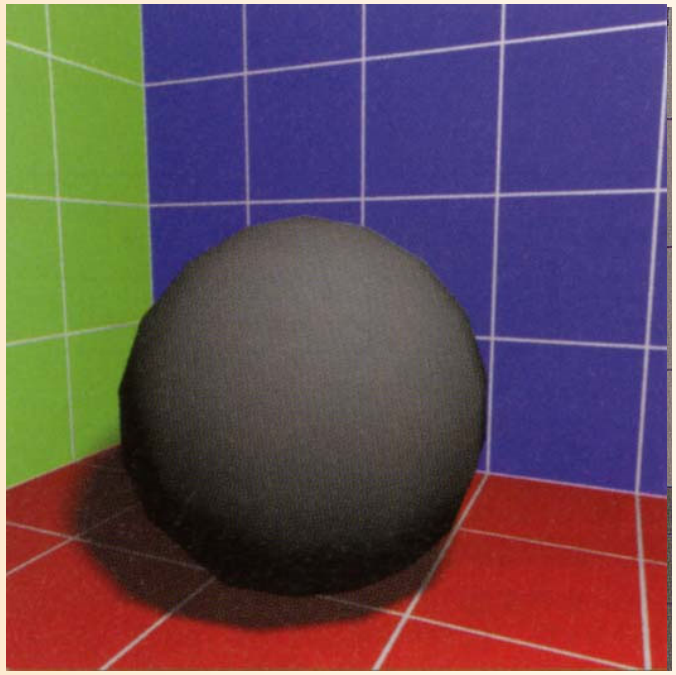
グローシェーディング



フォンシェーディング

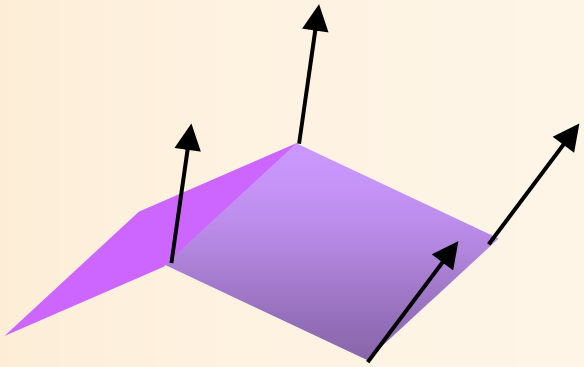
グローシェーディング

- グローシェーディング（Gouraud Shading）
  - 頂点の法線をもとに頂点の色を決定
  - 頂点の色をなめらかに補間して面内の色を計算
- 特徴
  - フラットシェーディングより
    なめらかに見える
  - 頂点の色を補間している
    だけなので、面内部の色は
    正しくない場合もある



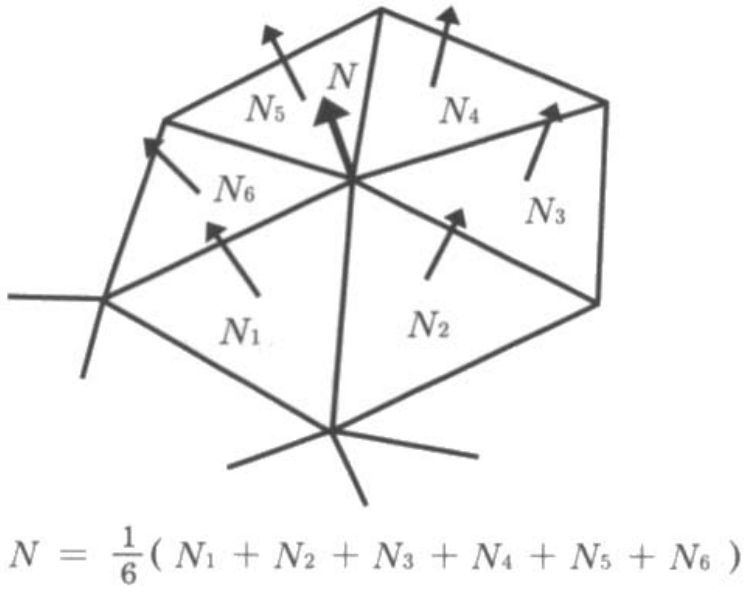
グローシェーディング

- グローシェーディングの手順
  - あらかじめ頂点の法線を計算しておく
    - 面の法線から計算（次のスライドで説明）
  - 頂点の法線をもとに頂点の色を決定
    - フラットシェーディングと同じ光源処理を適用
  - 頂点の色をなめらかに補間して、面内の各ピクセルの色を計算
  - 面ごとではなく頂点ごとに色を計算して補間するのがポイント



頂点の法線

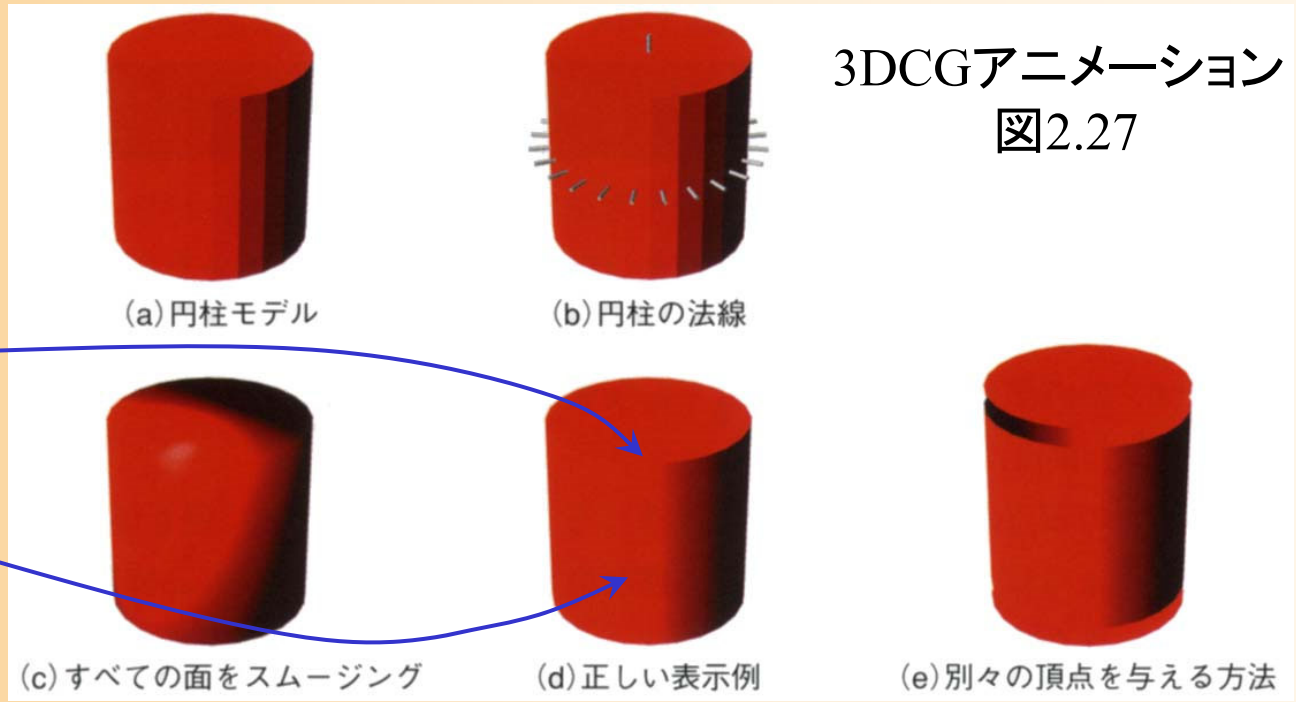
- 頂点の法線
  - もともと頂点には法線という概念はない
  - シェーディングを計算するために、頂点の法線を利用
- 計算方法
  - 頂点に隣接する全ての面の法線を平均
    - 面の面積に応じて加重平均する方法もある



基礎と応用 図4.2

面の法線と頂点の法線

- 頂点によっては面の法線を使用した方が良い場合もある
  - 同じ頂点でも隣接する面で異なる法線を使用



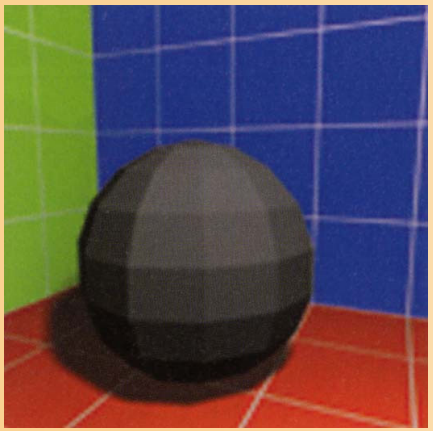
3DCGアニメーション
図2.27

面の法線（隣接する面同士で法線を共有せず）

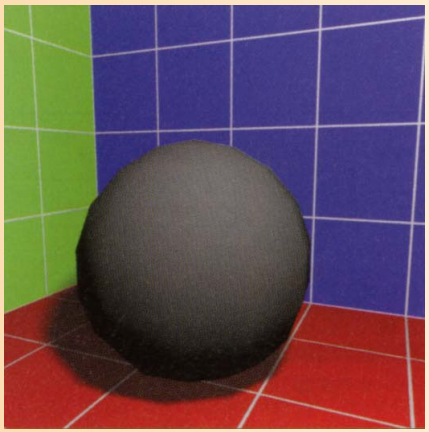
頂点の法線（隣接する面同士で法線を共有）

シェーディングの方法

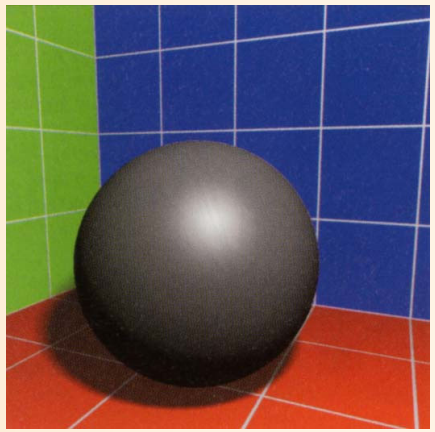
- フラットシェーディング
- スムーズシェーディング
  - グローシェーディング
  - フォンシェーディング



フラットシェーディング



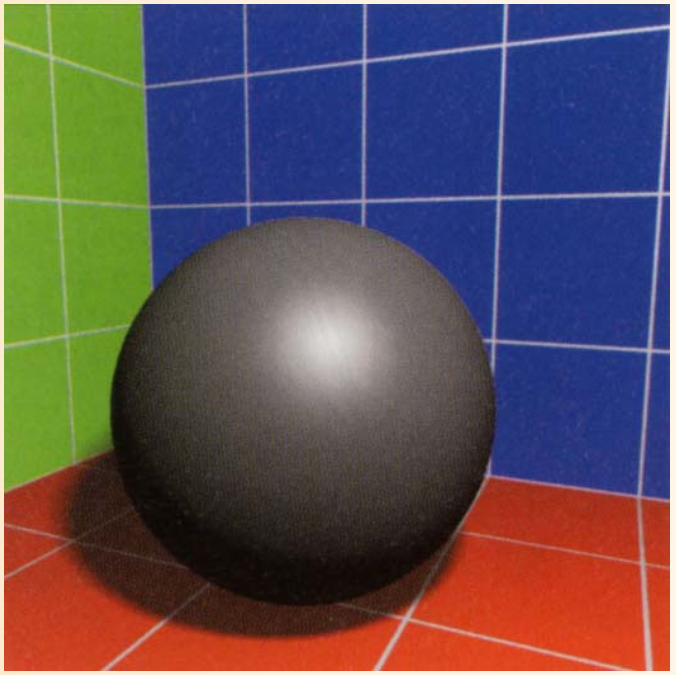
グローシェーディング



フォンシェーディング

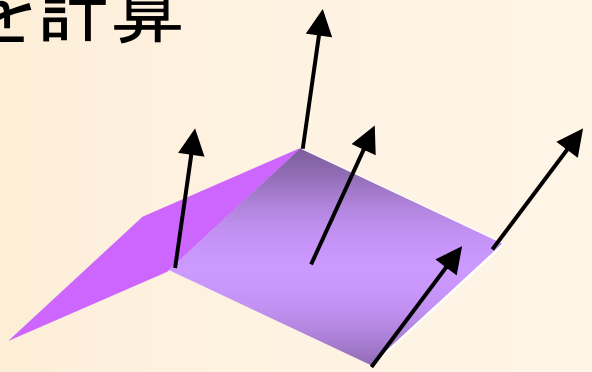
フォンシェーディング

- フォンシェーディング（Phong Shading）
  - 頂点の法線を補間して面内の各点の法線を決定
  - 面内の各点の法線にもとづき各点の色を計算
- 特徴
  - グローシェーディングでは表現することが難しかったハイライトも表現できる
  - 法線の正規化が必要になるため、計算時間がかなり余分にかかる



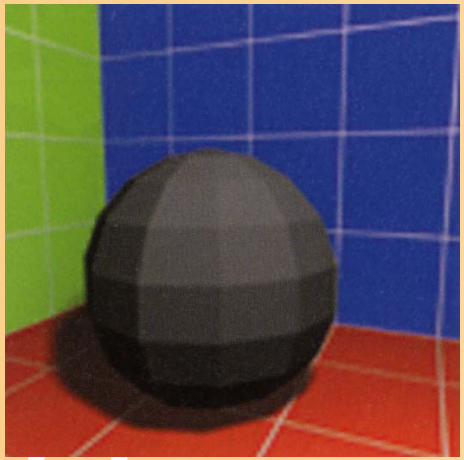
フォンシェーディング

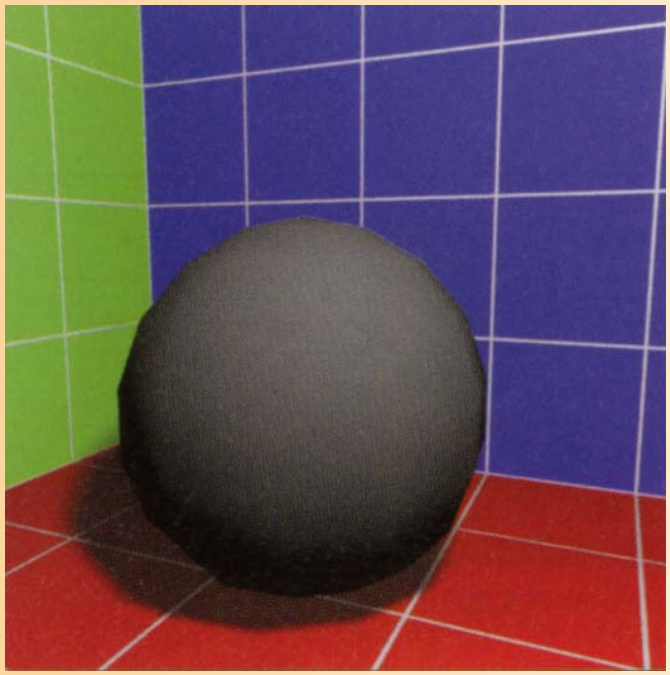
- フォンシェーディングの手順
  - グローシェーディングと同じく、頂点の法線をあらかじめ計算しておく
  - 頂点の法線を補間して面内の各ピクセルの法線を決定
  - 面内の各ピクセルの法線にもとづき、各ピクセルごとに光源処理を適用し、色を計算
  - 各ピクセルごとに光源計算を行なうのがポイント

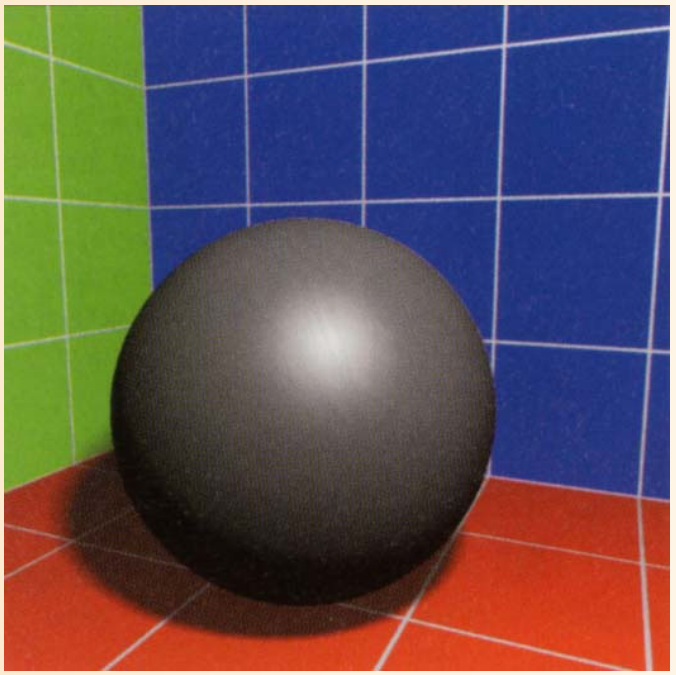


シェーディング結果の比較例

- グローとフォンシェーディングの比較
  - グローシェーディングでは消えているハイライトが表現されている点に注目







フォンシェーディングの処理時間

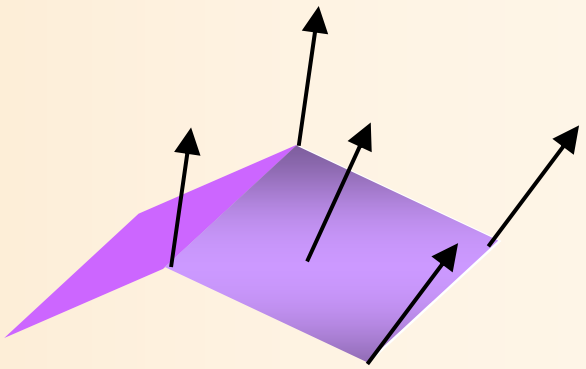
- フォンシェーディングには時間がかかる
  - 各ピクセルごとに、法線を計算する必要がある
    - 各頂点の法線を単純に補間しただけでは、法線の長さが 1 にならないため、正規化を行なう必要がある
    - 正規化を行なうためには、ベクトルの長さの計算（平方根の計算）を行なう必要があり、時間がかかる

$$\mathbf{N} = \sum_{i=1}^{n} w_i \mathbf{N}_i \quad \left(\sum_{i=1}^{n} w_i = 1 \right)$$

頂点とピクセルの距離により決まるウェイト w_i に応じて頂点の法線 $\mathbf{N}_i$ を補間

$$\mathbf{N}' = \frac{\mathbf{N}}{|\mathbf{N}|}$$

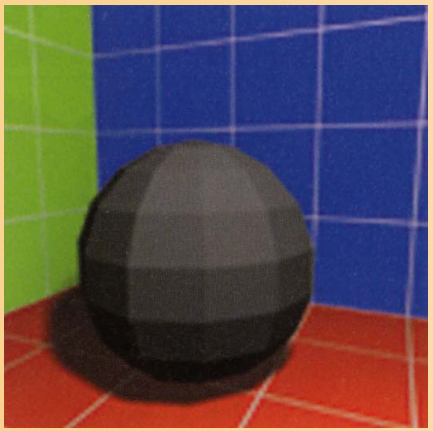
長さが 1 になるよう正規化



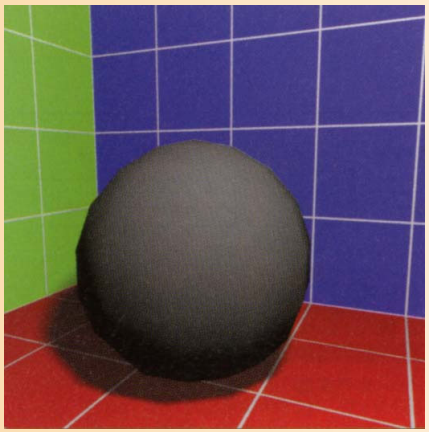
  - 各ピクセルで光源処理が必要

シェーディングの方法のまとめ

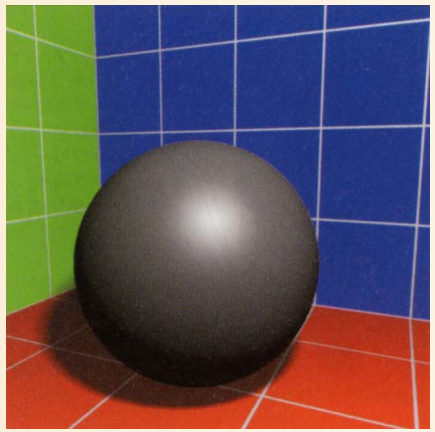
- フラットシェーディング
- スムーズシェーディング
  - グローシェーディング
  - フォンシェーディング



フラットシェーディング



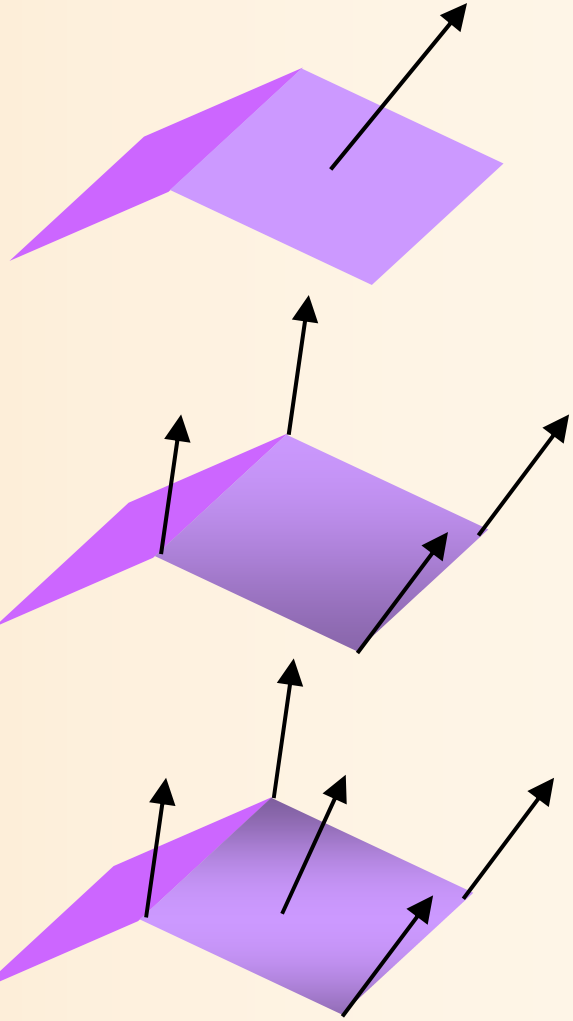
グローシェーディング



フォンシェーディング

シェーディングの処理のまとめ

- フラットシェーディング
  - 面の法線から面の色を計算
- グローシェーディング
  - 頂点の法線から頂点の色を計算
  - 頂点の色から、各点の色を決定
- フォンシェーディング
  - 頂点の法線から、面内の各点（ピクセル）の法線を計算
  - 各点の法線から、色を計算



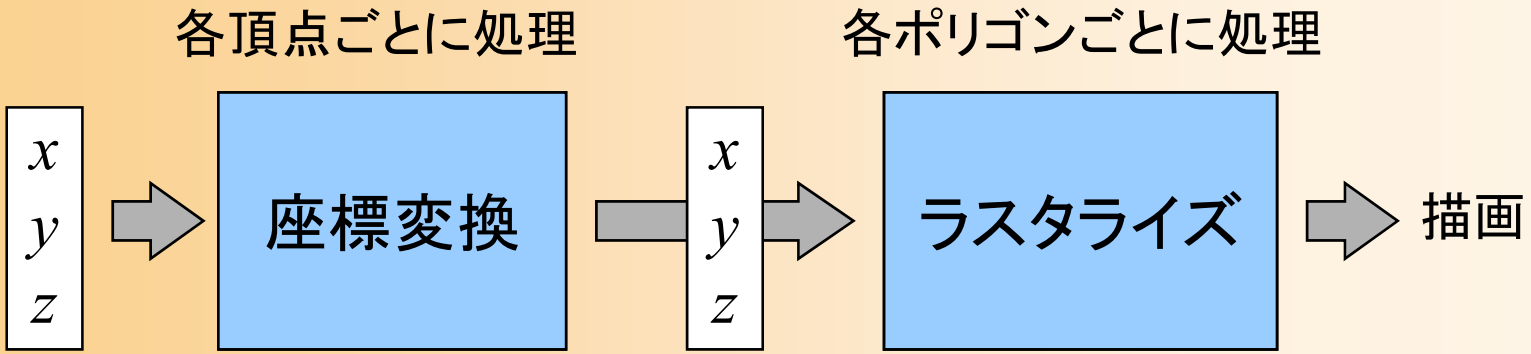
シェーディング方法の応用

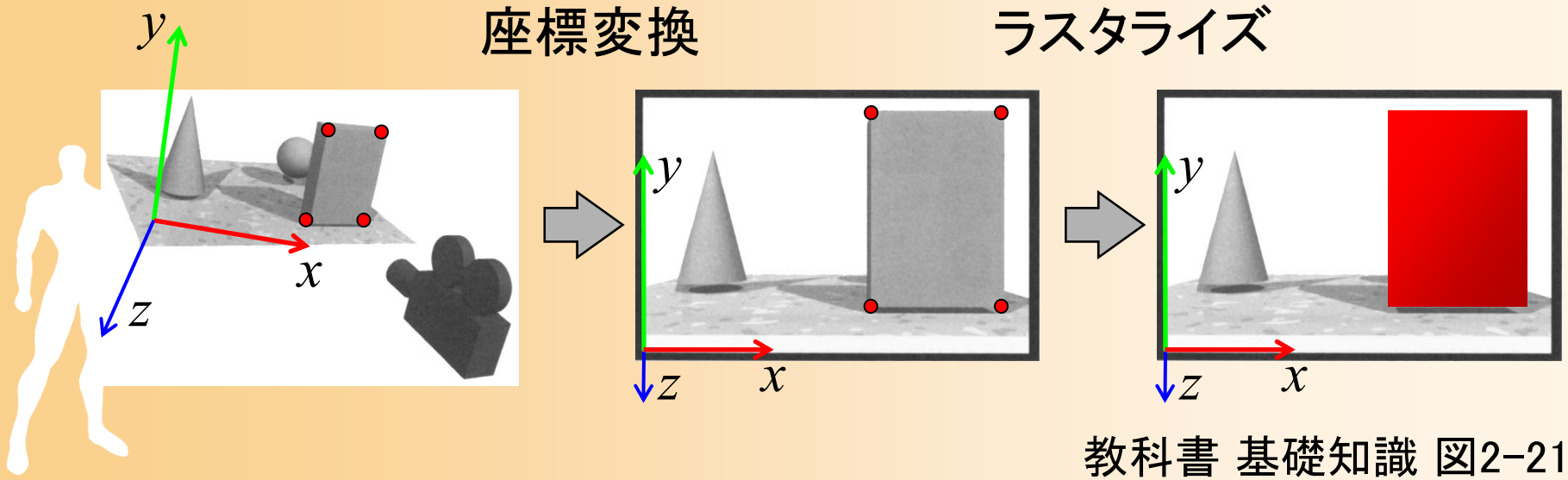
- フラットシェーディング
  - 頂点色の補間がハードウェアでできなかった昔はよく使われていた
  - ポリゴンらしさを表現する時にあえて使われることがある
- グローシェーディング
  - 現在、主に使われている方法
  - ハードウェアでサポート（ソフトでも十分実現可）
  - 実際には次に述べるマッピング技術と組み合わせて使われることが多い
- フォンシェーディング
  - 時間がかかるので、速度が要求される用途にはあまり向かない（最近は、PixelShaderにより高速に実現可能）

プログラマブル・シェーダ（1）

- プログラマブル・シェーダ
  - ごく最近のグラフィックカードが持つ新機能
  - 少し前までは、シェーディングの処理は基本的にハードウェアで固定の処理として実行されていた
  - シェーディングの方法をプログラマが変更可能にすることで、独特の効果などを表現することが可能になる
- 実現方法
  - シェーディングの計算は、グラフィックカード上の演算装置（GPU）で実行される
  - 独自のシェーディング処理を専用言語で記述・コンパイルしておき、実行時にGPUに送る

レンダリング・パイプライン（復習）



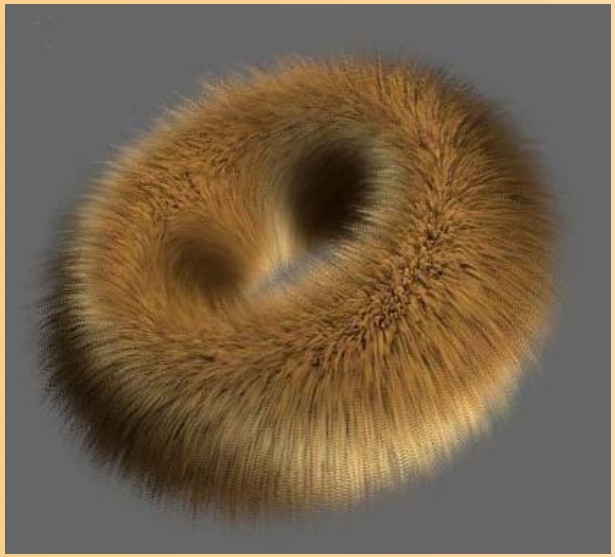


教科書 基礎知識 図2-21

プログラマブル・シェーダ（2）

- 主なプログラマブル・シェーダの種類
  - Vertex Shader
    - 頂点の位置・色を計算する処理を記述
  - Pixel Shader
    - 面内の各点の色を計算する処理を記述
- 専用言語
  - 最初はアセンブラ風の単純な言語だけだった
  - C言語に近い言語（cg）が最近になって登場
- 近年は、レンダリング以外の計算処理にも広く用いられている（GPGPU）

プログラマブル・シェーダの応用例



毛の描画 [Kano]

錆の描画 [nVidia]

肌の描画 [nVidia]

- 現在、最も注目されている技術のひとつ
  - 今までリアルタイムでは困難だった特殊な描画や高品質の画像を実現できる

まとめ

- レポート課題

- シェーディング
- 光のモデル
- スムーズシェーディング

次回予告

- 次回（講義）
  - シェーディング（続き）
    - OpenGLでの光源情報の設定
    - ラジオシティ
    - 影の表現
    - BRDF
  - マッピング
- 次々回（OpenGL演習）
  - シェーディング、マッピング