コンピューターグラフィックスS

第13回　演習(4):シェーディング、マッピング

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･前々回の復習

- シェーディング（光源設定の変更）
- テクスチャマッピング

前回･前々回の復習

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）

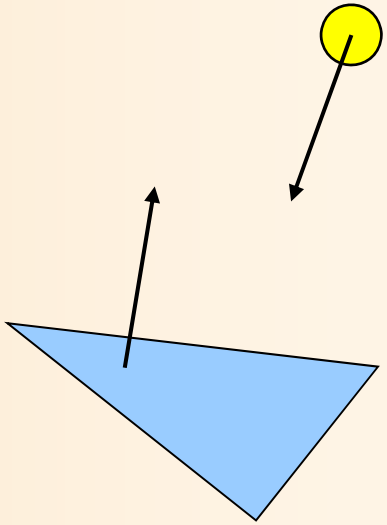
$$I = I_a k_a + \sum_{i=1}^{n_L} I_i \left[k_d \left(N \cdot L \right) + k_s \left(R \cdot V \right)^n \right] + k_r I_r + k_t I_t$$

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

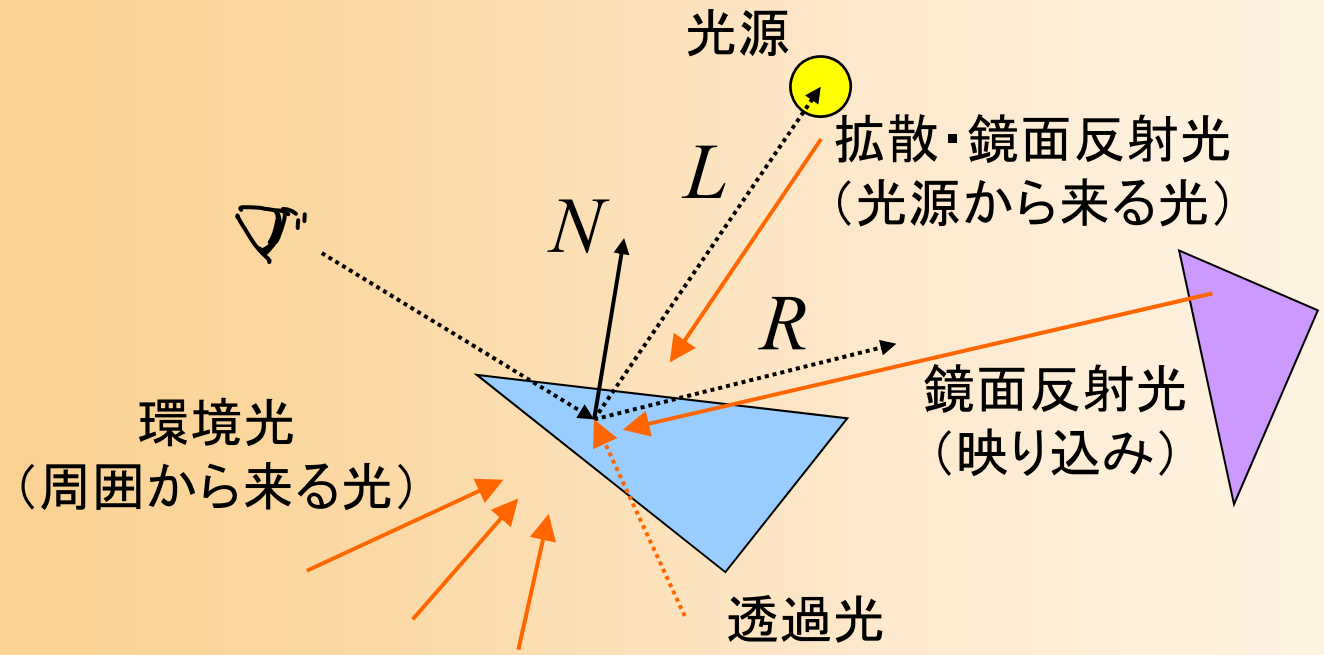
- 局所照明モデル
  - 光源と一枚の面の関係のみを考慮したモデル
    - 環境光、拡散反射光、鏡面反射光



- 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（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）

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

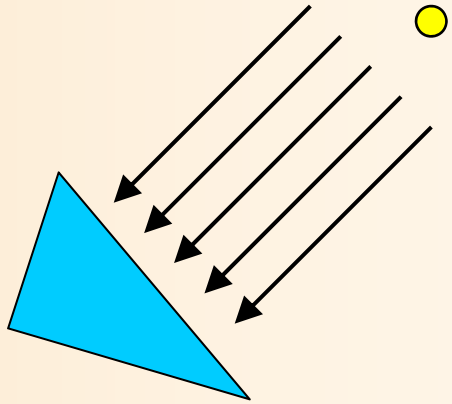
環境光　拡散反射光　鏡面反射光（局所照明）　鏡面反射光（大域照明）　透過光

それぞれの光源からの光（局所照明）　大域照明

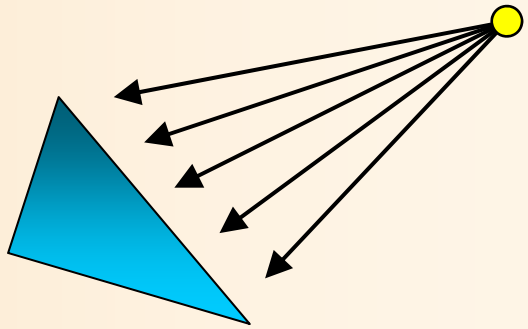
光源の種類

- 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

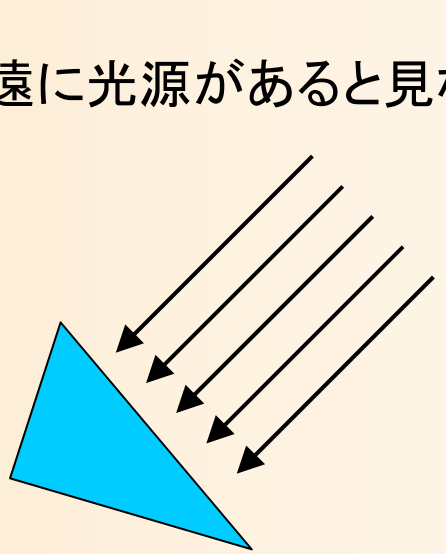


- 点光源
  - 位置の決まった光源
  - ライトなどの表現に適している
  - 光の方向は点光源と面の位置関係により決まる
  - 光の減衰も考慮できる

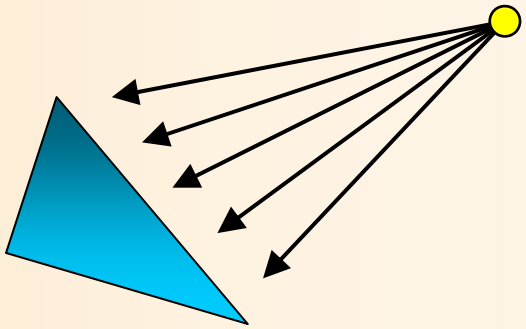


光源の種類と設定方法

- 平行光源
  - (x,y,z)の方向から平行に光が来る
  - 光源位置のw座標を0.0に設定



- 点光源
  - (x,y,z)の位置に光源がある
  - 光源位置のw座標を1.0に設定



光源情報の設定の例

- 光源の位置や色の設定
  - 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 );
```

光源情報の設定の例

- サンプルプログラムの例

```
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 );
```

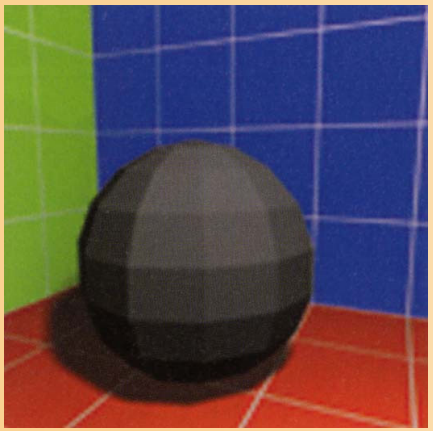
光源位置のw座標が1.0なので、点光源となる

LIGHT0の
・光源の位置・種類
・拡散反射成分の色
・鏡面反射成分の色
を設定

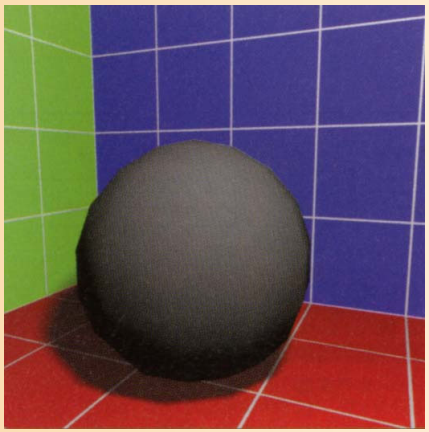
LIGHT0の
・環境光成分の色
を設定

シェーディングの方法

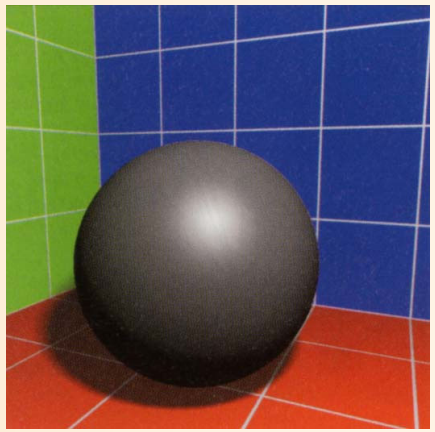
- フラットシェーディング
- スムーズシェーディング
  - グローシェーディング
  - フォンシェーディング



フラットシェーディング



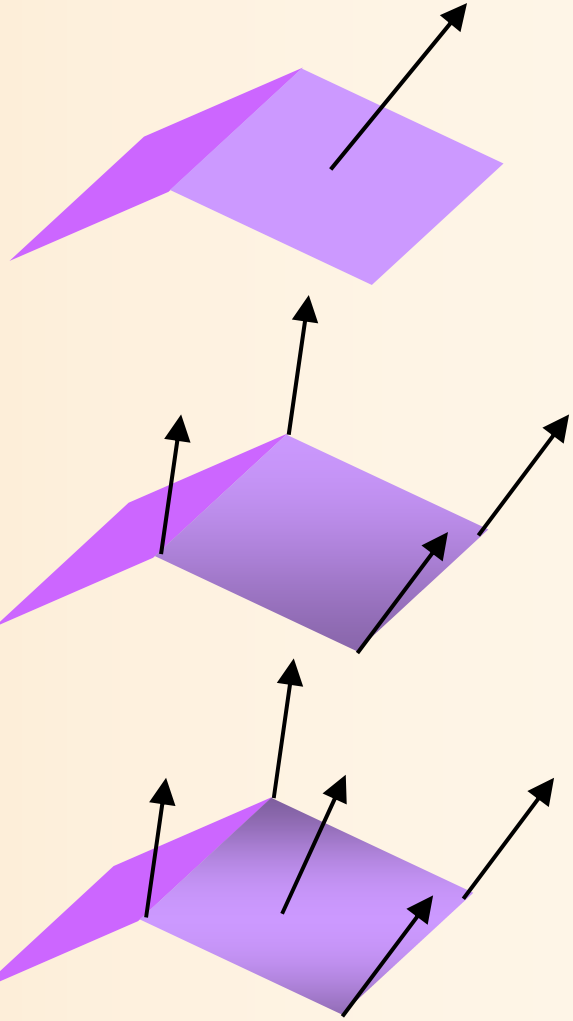
グローシェーディング



フォンシェーディング

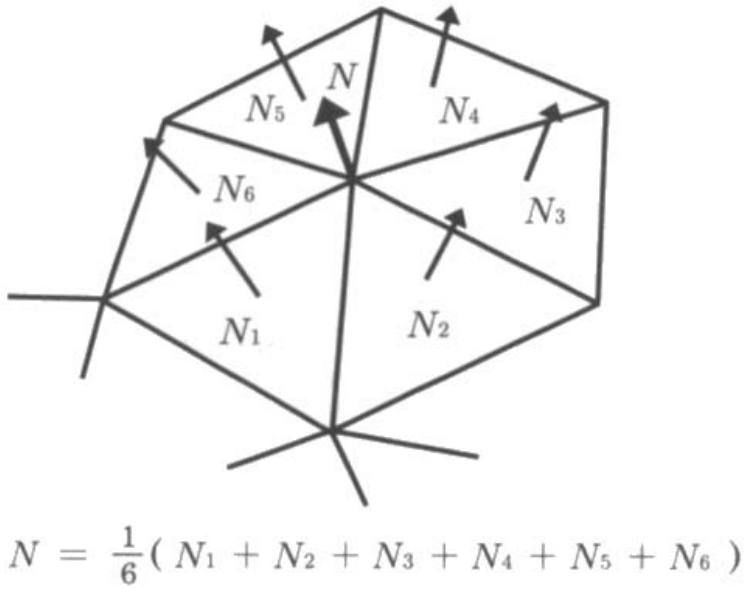
シェーディングの処理のまとめ

- 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



頂点の法線

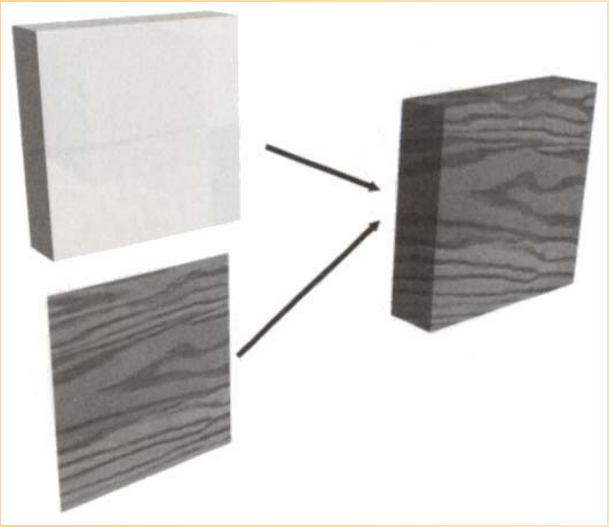
- 頂点の法線
  - もともと頂点には法線という概念はない
  - シェーディングを計算するために、頂点の法線を利用
- 計算方法
  - 頂点に隣接する全ての面の法線を平均
    - 面の面積に応じて加重平均する方法もある



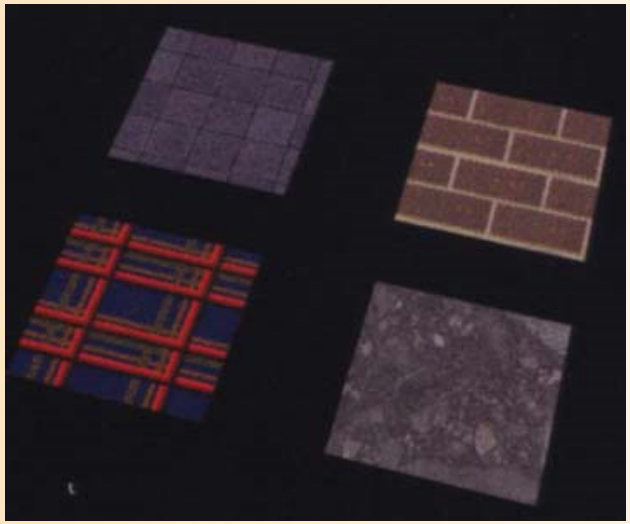
基礎と応用 図4.2

テクスチャマッピング

- マッピング
  - 面を描画する時に、面の表面に画像を貼り付ける技術
  - 複雑なモデリングをすることなく、細かい模様などを表現できる



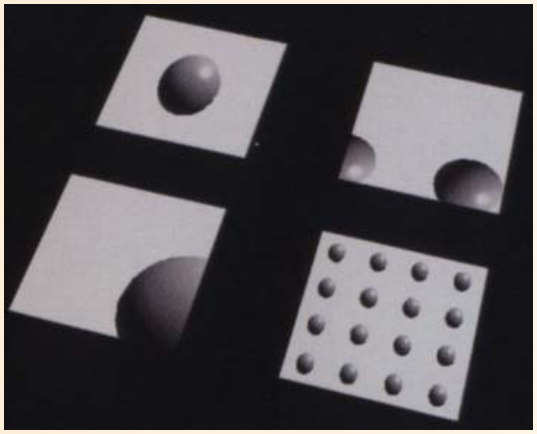
基礎知識 図3-19



基礎と応用
図5.2

マッピングの方法

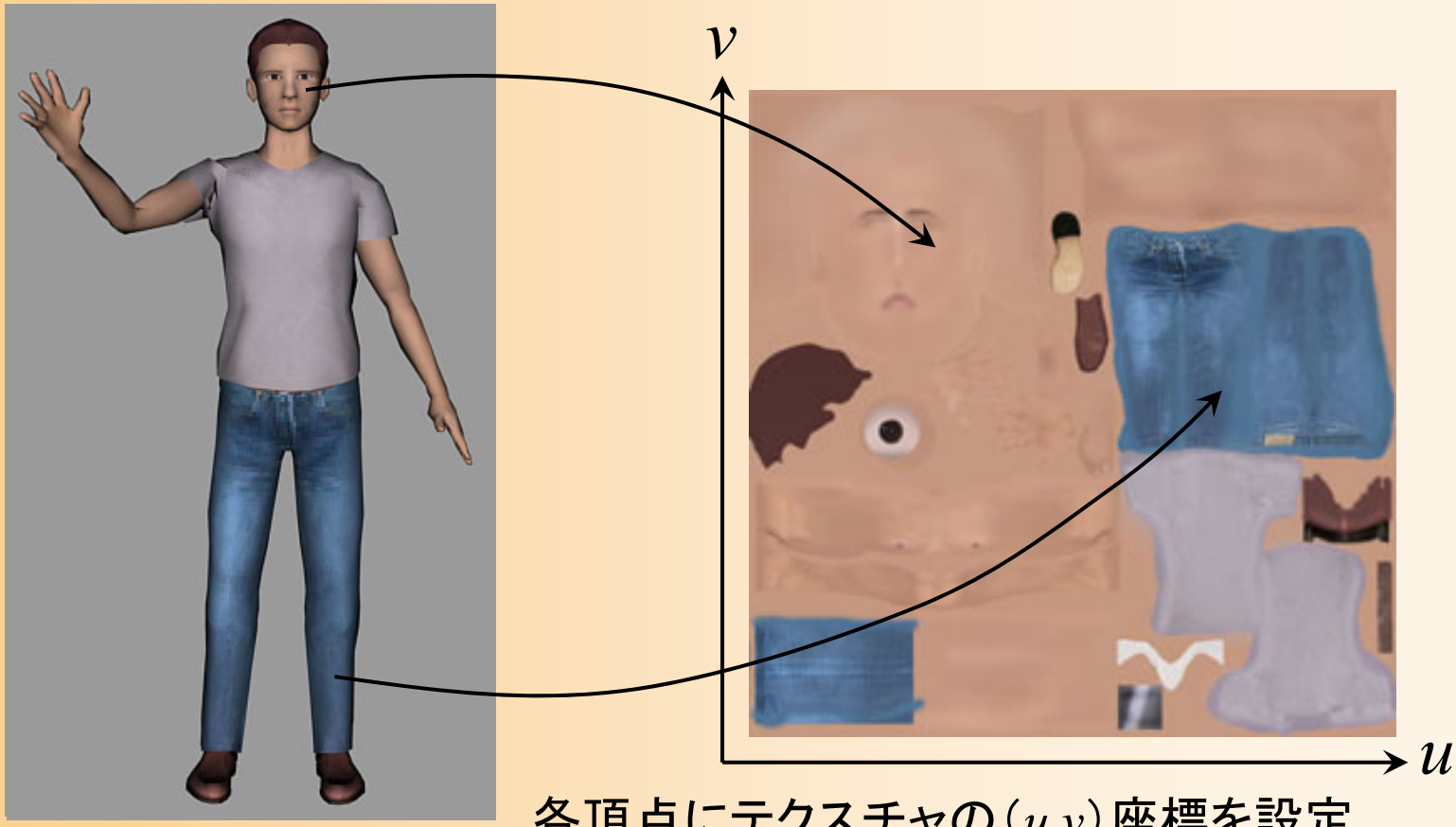
- 物体への画像の貼り付け方
  - マッピングの方向や繰り返しの方法
- uv座標系
  - テクスチャ画像の座標は(u,v)で表せる
  - モデルデータの各頂点(x,y,z)ごとに、対応するテクスチャ画像の(u,v)座標を与えておく



基礎と応用 図5.3

マッピングの例

- 人体モデルへのマッピングの例



各頂点にテクスチャの(u,v)座標を設定

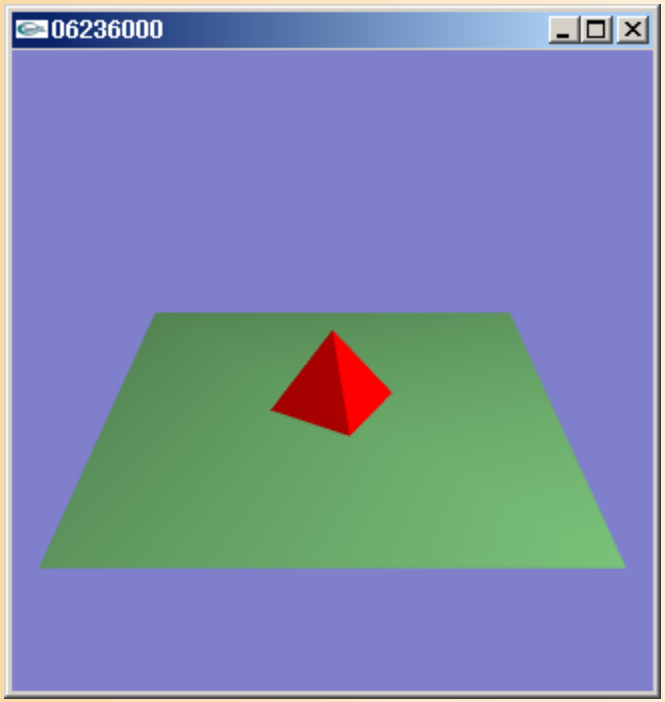
光源設定の変更

光源設定の変更

- 点光源の位置を変更
- 地面の描画を変更
- 点光源から平行光源へ変更
- 点光源の位置を変更

演習準備

- 以前作成した、四角すいの描画関数を使って、ひとつの四角すいを描画するように、描画処理を変更



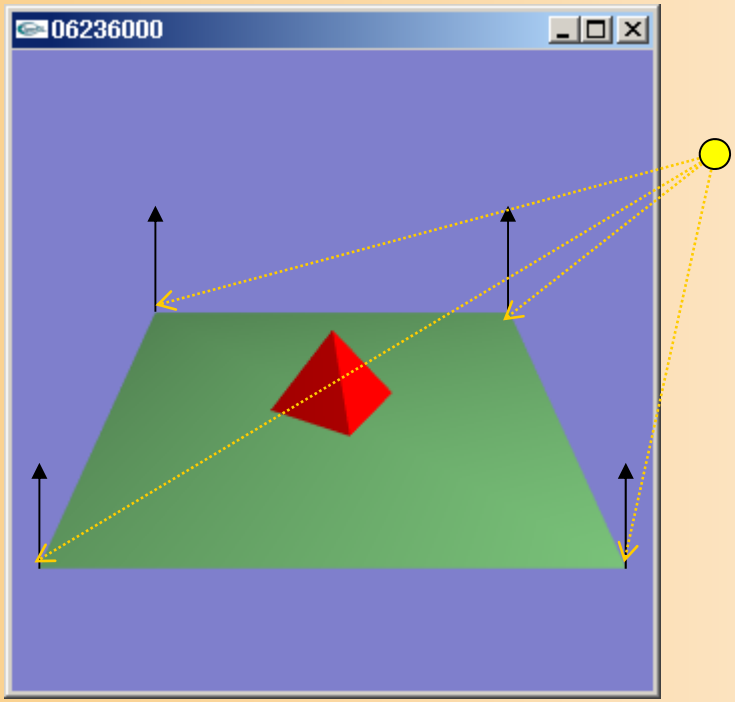
1. 点光源の位置を変更

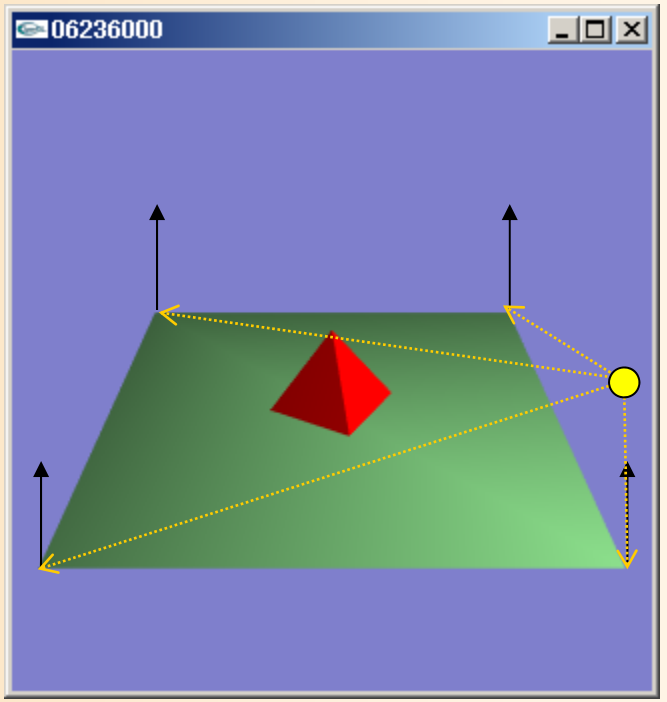
- 点光源の位置を変更してみる
  - 描画関数の光源の位置を変更
    - 初期化関数で設定した値は、すぐに描画関数で更新されるので、初期化関数は無理に修正する必要はない
  - 点光源の効果が分かるように、地面に近づける

```
void  display( void )
{
        ・・・・・・
        // 光源位置を設定（変換行列の変更にあわせて再設定）
        float  light0_position[] = { 5.0, 3.0, 5.0, 1.0 };
        glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
        ・・・・・・
```

点光源の影響

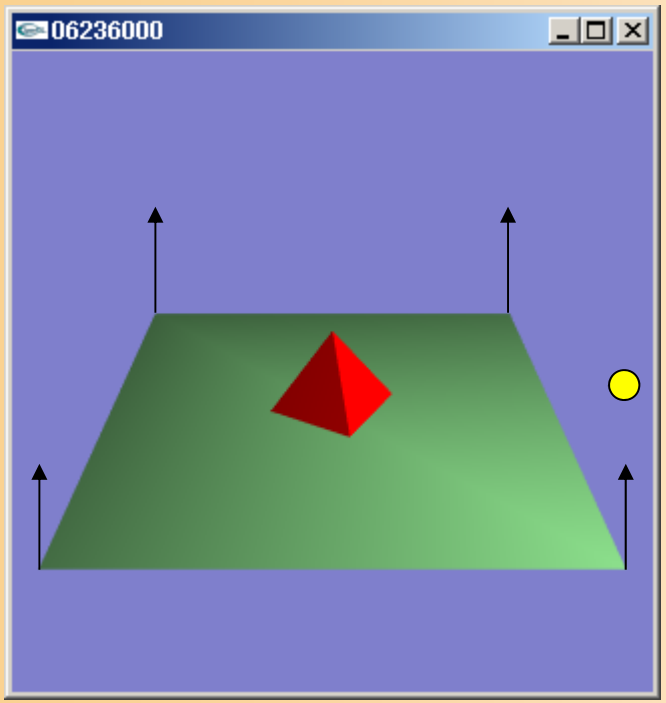
- 点光源(10,10,10)
  - 各頂点における、光源ベクトルと法線の角度が小さい
- 点光源(5,3,5)
  - 光源が近いので、光源ベクトルと法線の角度が大きくなる





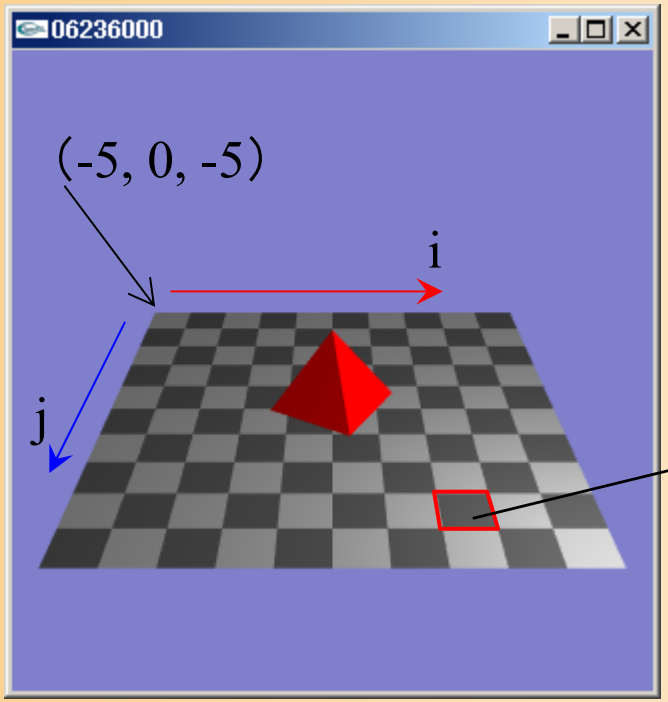
2. 地面の描画を変更

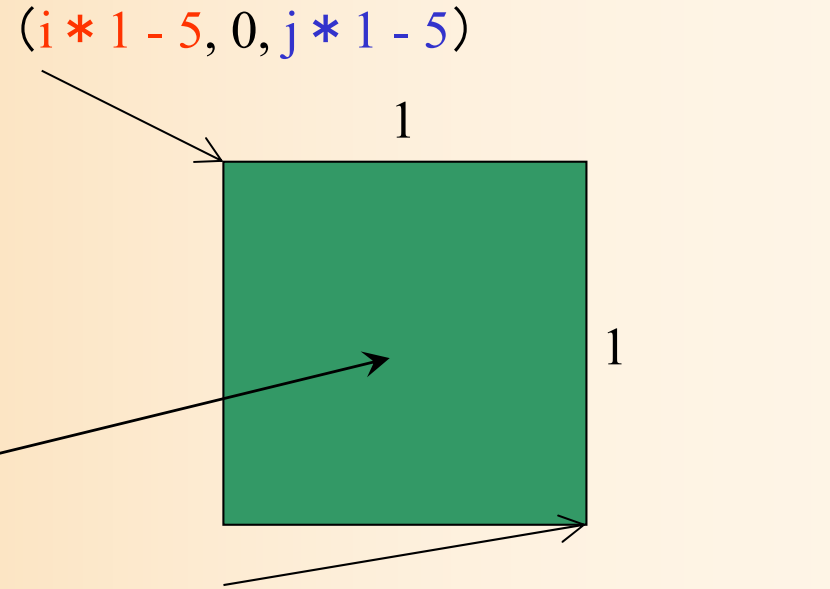
- 点光源の効果がより分かるように、地面を100枚の四角形に分けて描画してみる
  - 各頂点ごとに光源計算が行われるため、より正確



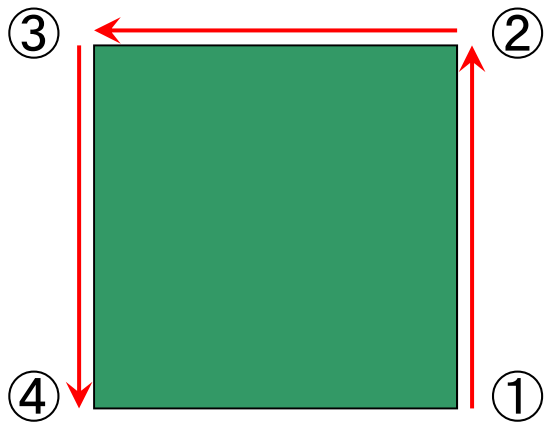
地面の描画

- 10×10の四角形を並べて描画
  - 全体の幅を10とすると、各四角形の幅は1となるので、４点の座標が計算できる



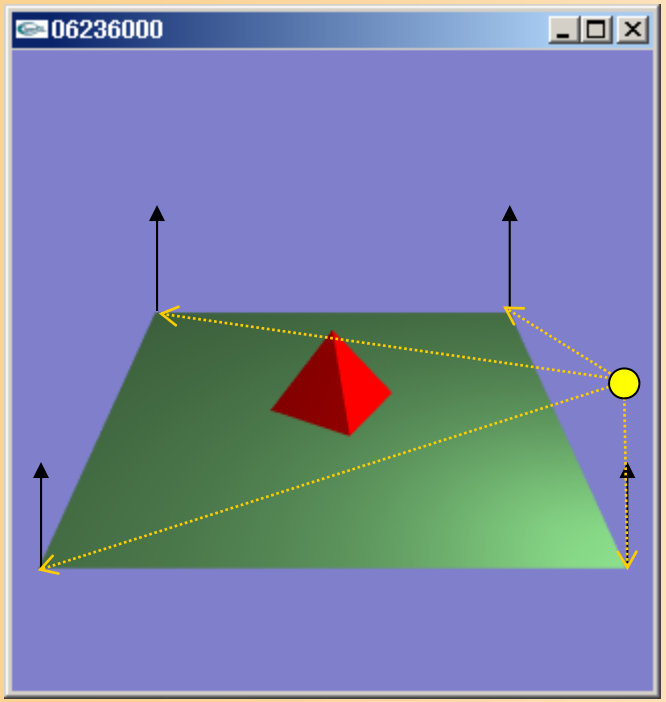


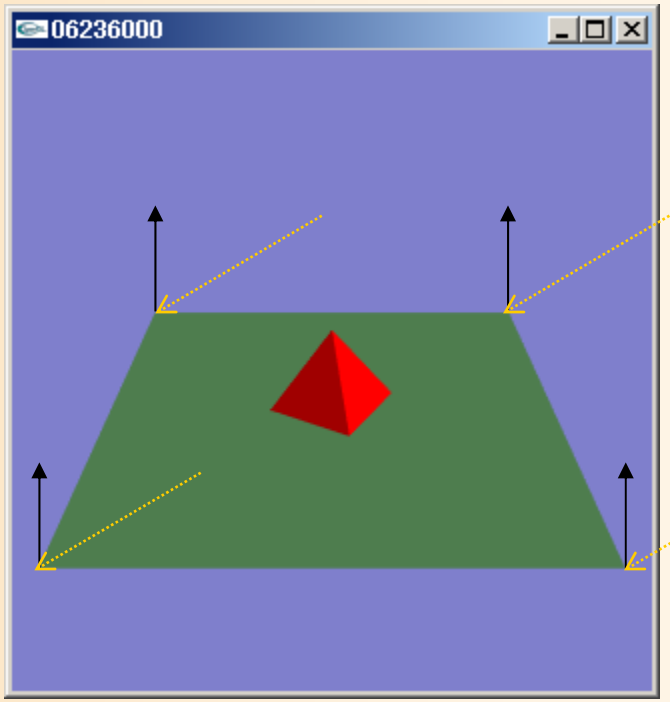
```
// 地面を描画
int  i, j;
glBegin( GL_QUADS );
        glNormal3f( 0.0, 1.0, 0.0 );
        glColor3f( 0.5, 0.8, 0.5 );
        for ( i=0; i<10; i++ )
        {
                for ( j=0; j<10; j++ )
                {
                        glVertex3f( i - 4.0, 0.0, j - 4.0 );    ①
                        glVertex3f( i - 4.0, 0.0, j - 5.0 );    ②
                        glVertex3f( i - 5.0, 0.0, j - 5.0 );    ③
                        glVertex3f( i - 5.0, 0.0, j - 4.0 );    ④
                }
        }
glEnd();
```



4. 点光源から平行光源へ変更

- 平行光源に設定して、点光源との違いを確認してみる
  - 平行光源では、全頂点の光源方向が同じになる





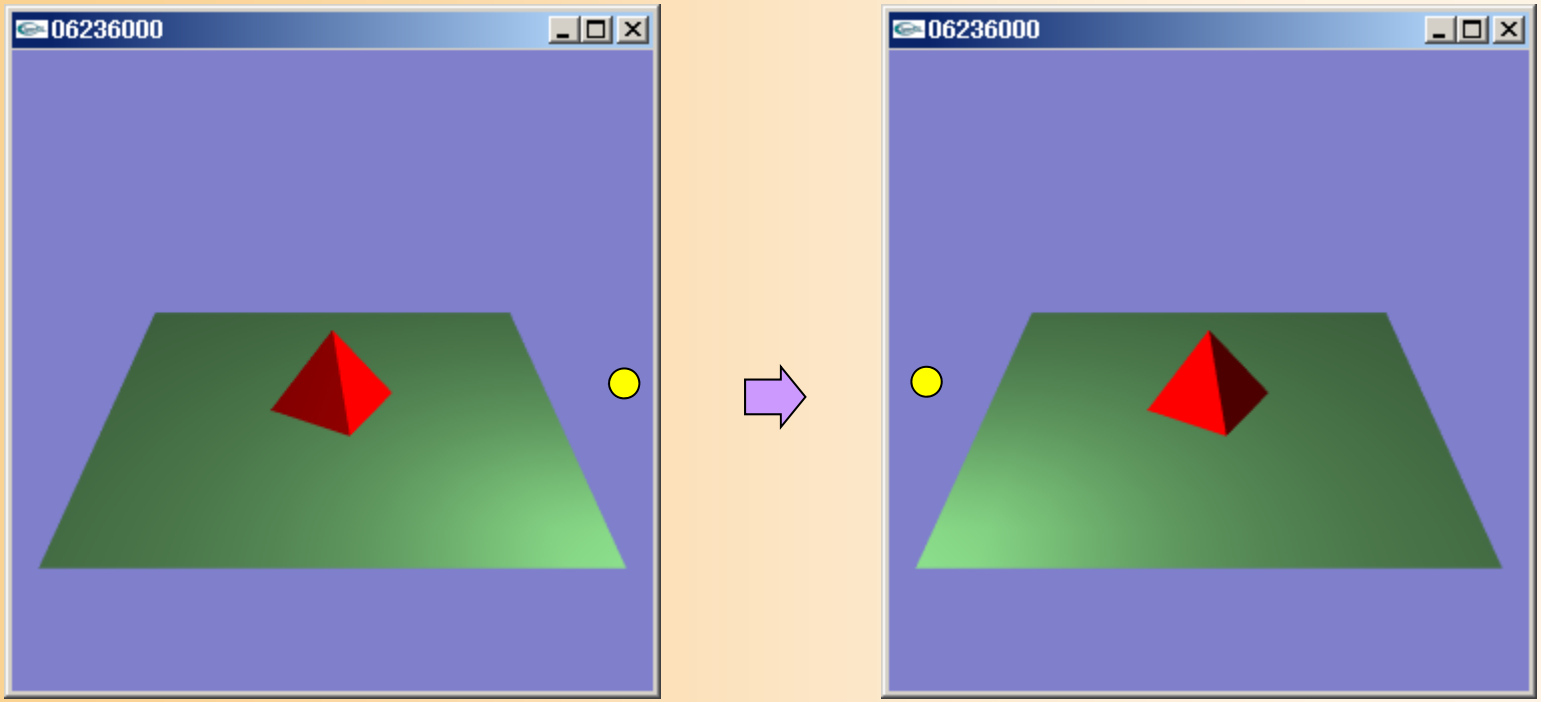
光源の種類の変更

- 平行光源に設定
  - 光源位置の w座標を 0.0 に指定

```
void  display( void )
{
        ……
        // 光源位置を設定（変換行列の変更にあわせて再設定）
        float  light0_position[] = {5.0, 3.0, 5.0, 0.0 };
        glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
        ……
}
```

演習課題(1)

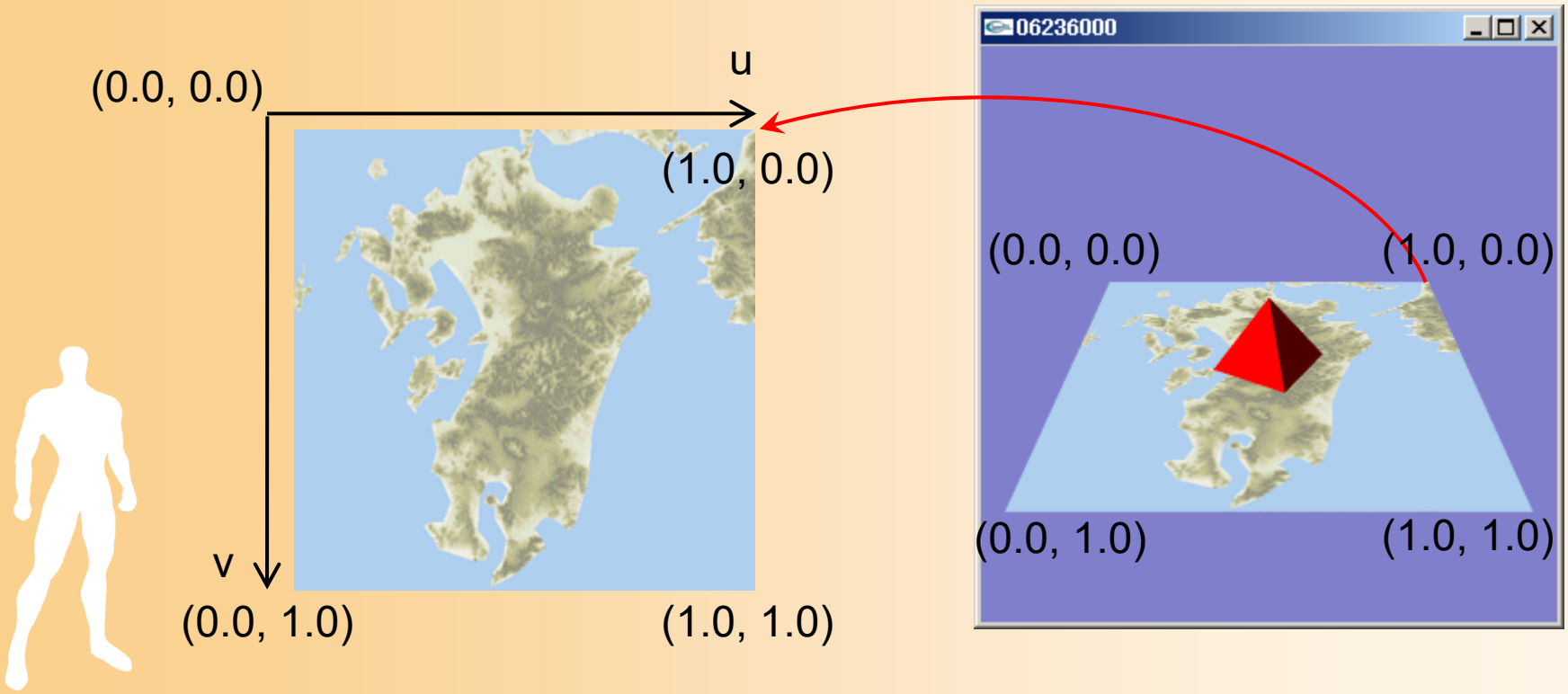
- 図のように、左下手前が照らされるように、光源位置の設定を修正してみよ



テクスチャマッピング

テクスチャマッピング

- 地面にテクスチャマッピングを適用して描画
  - 各頂点に、テクスチャ座標(u,v)を指定

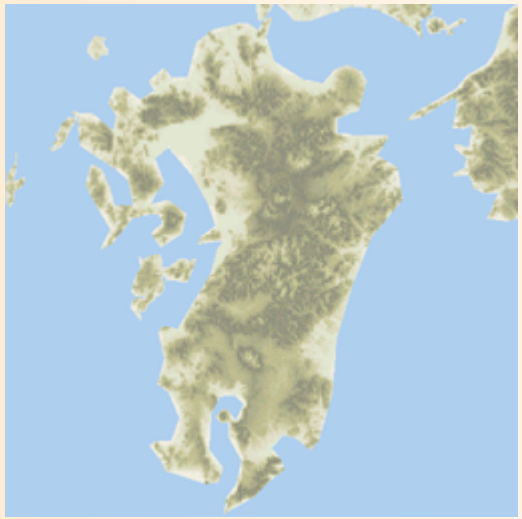


プログラムの修正

- プログラムの修正箇所
  - テクスチャ画像を格納する変数を追加
  - テクスチャ画像の読み込み処理、テクスチャマッピングの設定処理、を追加
    - 画像の読み込みには bitmap.cpp の関数を使用
  - 地面にテクスチャマッピングを行う処理を追加
    - テクスチャマッピングの有効化
    - 各頂点にテクスチャ座標を設定
- コンパイル方法の変更
  - bitmap.cpp を一緒にコンパイル

テクスチャ画像

- サンプルのテクスチャ画像
  - 本演習用の画像ファイル（kyushu.bmp）
  - 24ビットBMP画像
  - 256×256ピクセル

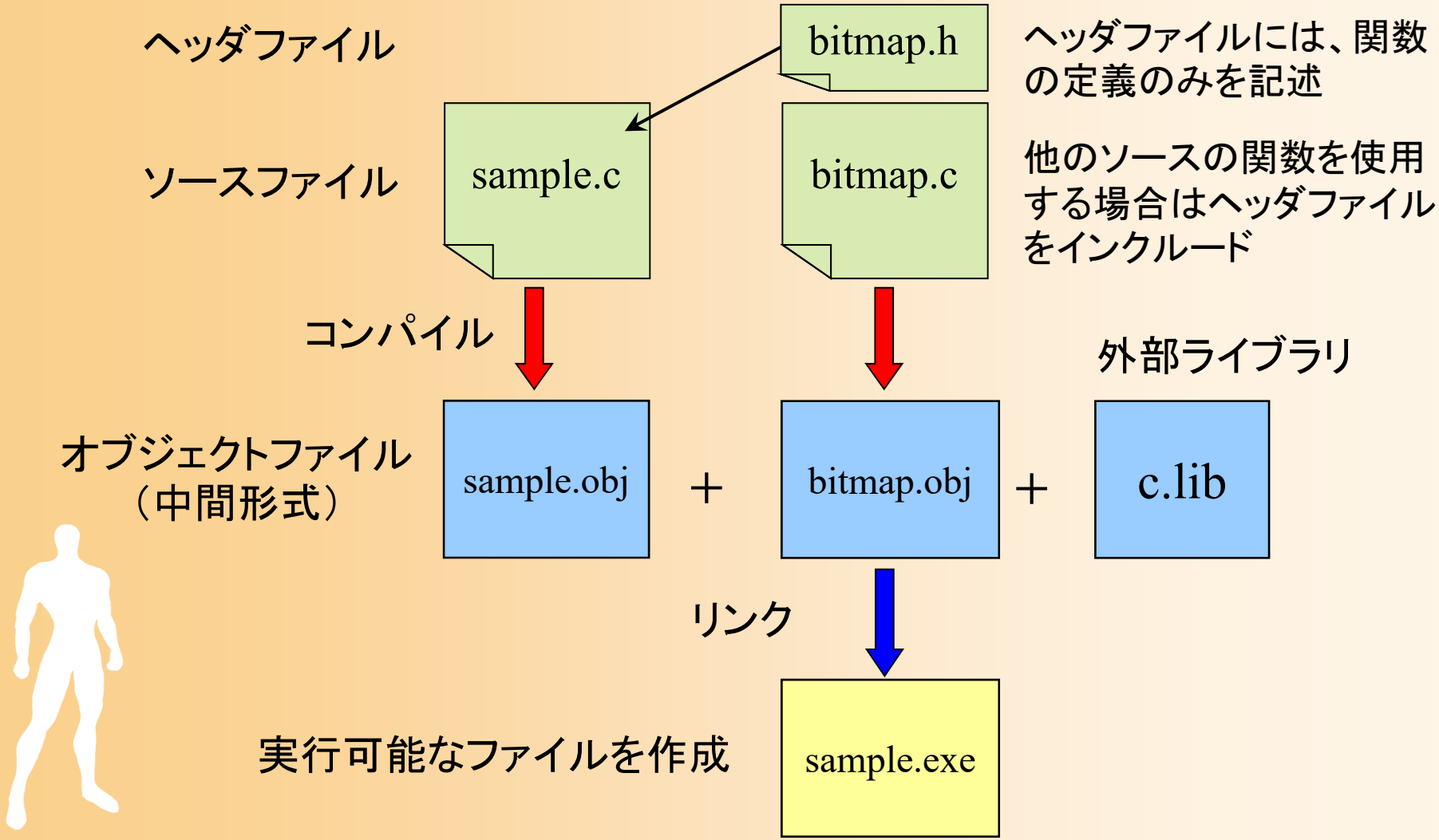


- Moodleに置かれている画像ファイルを、プログラムから読み込める場所に配置する
  - 実行ディレクトリの設定を変更していなければ、プロジェクトファイルと同じディレクトリに配置

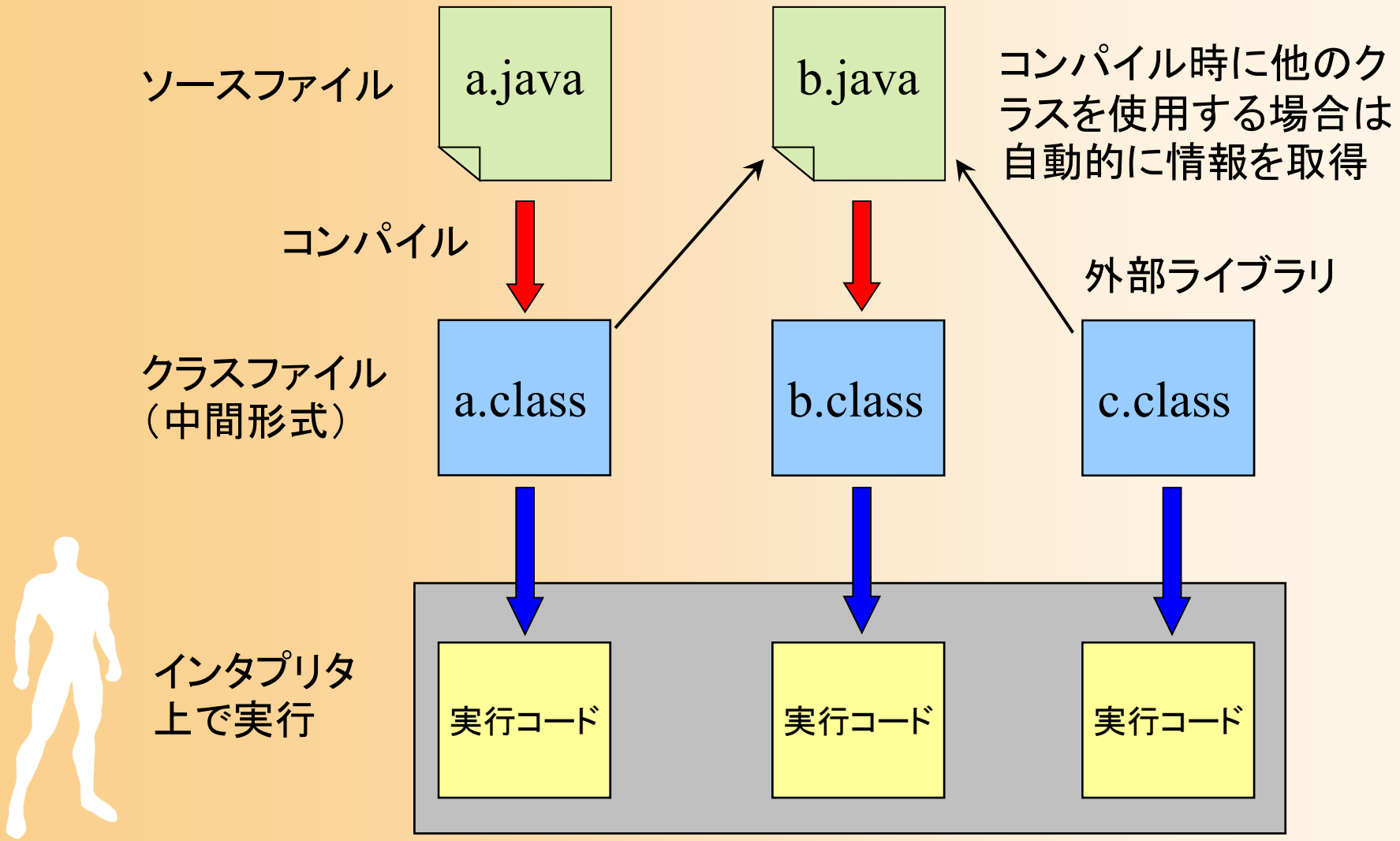
画像の読み込み関数の利用(1)

- テクスチャマッピングを行うためには、通常、画像をファイルから読み込む必要がある
  - C言語の標準ライブラリには、画像を読み込む機能は無いので、自分で作成する必要がある
- サンプルプログラムとして、画像の読み込み関数を含むファイル(bitmap.h, bitmap.cpp)を用意しているので、これらを利用する

C言語のコンパイル・リンクの仕組み



Javaのコンパイルと実行の仕組み（参考）



画像の読み込み関数の利用(2)

- ヘッダファイルのインクルードの追加
  - bitmap.h は、画像読み込み関数の定義が記述されている(関数を呼び出すために必要)

```
// Bitmap読み込み関数のためのヘッダファイルのインクルード
#include "bitmap.h"
```

- ソースファイルをプロジェクトに追加
  - 過去の演習のコンパイル方法の資料を参照
  - bitmap.h と bitmap.cpp の両方のファイルをプロジェクトに追加する

参考:Linux環境でのコンパイル方法

- コンパイル方法
  - bitmap.cpp をコンパイル時の引数に入力に追加
  - 複数のソースファイルのコンパイルとリンクが自動的に行われる

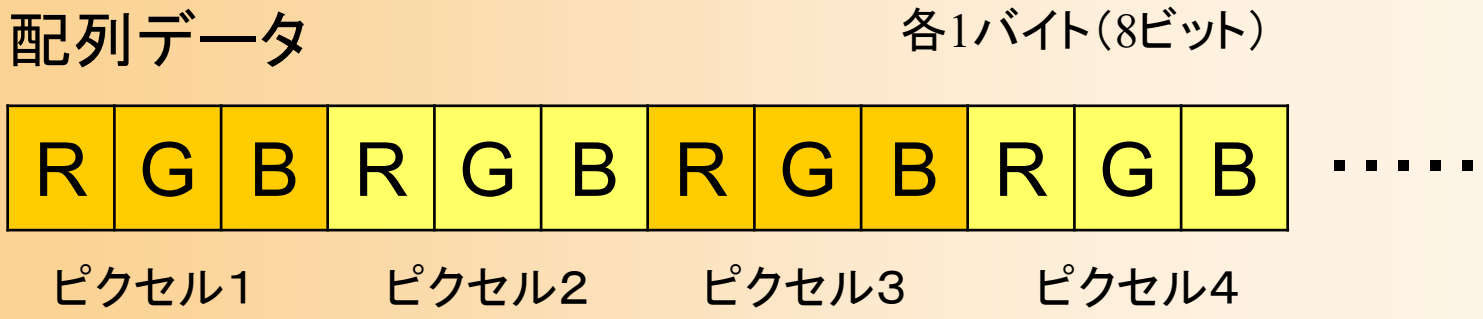
```
gcc opengl_sample.cpp bitmap.cpp -L/usr/X11R6/lib
-lglut -lGLU -lGL -lXmu -lm -o opengl_sample
```

テクスチャマッピングの手順

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

1. テクスチャ画像の読み込み

- 画像をファイルから配列に読み込む
  - OpenGLやC言語は、標準では、画像の読み込みはサポートしていないので、別のライブラリや、自作関数を使用する必要がある
  - 今回の演習では、サンプルプログラムとして用意された、24ビットBMP画像を配列に読み込む関数を利用する



テクスチャ画像読のみ込み

- テクスチャ画像を格納する変数を追加

```
// テクスチャ画像データ
int      tex_width;
int      tex_height;
unsigned char *   tex_image = NULL;
```

- 初期化処理に、画像を読み込む処理を追加

```
void  initEnvironment( void )
{
        ......
        // テクスチャ画像の読み込み
        loadBitmap( "kyushu.bmp", &tex_image,
                                  &tex_width, &tex_height );

        if ( tex_image == NULL )
                return;
```

画像のファイル名を指定（入力）

読み込んだ画像データの領域のアドレス（出力）

読み込んだ画像データのサイズ（出力）

2. テクスチャ画像の登録

- テクスチャマッピングを行うために、配列に格納された画像データを登録する
- glTextureImage2D()関数
  - 画像データの形式、サイズなどを指定
    - GL_RGB・・・RGB 形式
    - GL_RGBA・・・RGB＋不透明度(A) 形式
    - GL_UNSIGNED_BYTE・・・各色ごとに1バイト
  - 画像データを設定

3. テクスチャマッピングのパラメタ設定

- glTexParameter()関数
  - GL_TEXTURE_MAG_FILTER, GL_TEXTURE_MIN_FILTER
    - テクスチャの拡大縮小方法の設定
    - GL_NEAREST・・・一番近くのピクセルのみ使用
    - GL_LINEAR・・・近くのピクセルの色を補間して使用
  - GL_TEXTURE_WRAP_S, GL_TEXTURE_WRAP_T
    - テクスチャの繰り返しを行うかどうかの設定
    - GL_REPEATE or GL_CLAMP

テクスチャ画像の登録と設定

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

        // テクスチャ画像の読み込み
        loadBitmap( "kyushu.bmp", &tex_image, &tex_width, &tex_height );
        if ( tex_image == NULL )
                return;

        // テクスチャオブジェクトの設定
        glTexImage2D( GL_TEXTURE_2D, 0, GL_RGB, tex_width, tex_height, 0,
                GL_RGB, GL_UNSIGNED_BYTE, tex_image );

        // テクスチャマッピングの方法を設定
        glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT );
        glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT );
        glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
        glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
}
```

4. テクスチャマッピングの適用方法

- テクスチャをどのように適用するかを設定
- glTexEnv()関数
  - glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, 適用方法);
  - 適用方法には、以下のいずれかを指定
    - GL_REPLACE
    - GL_DECAL
    - GL_MODULATE
    - GL_BLEND

テクスチャマッピングの適用方法(1)

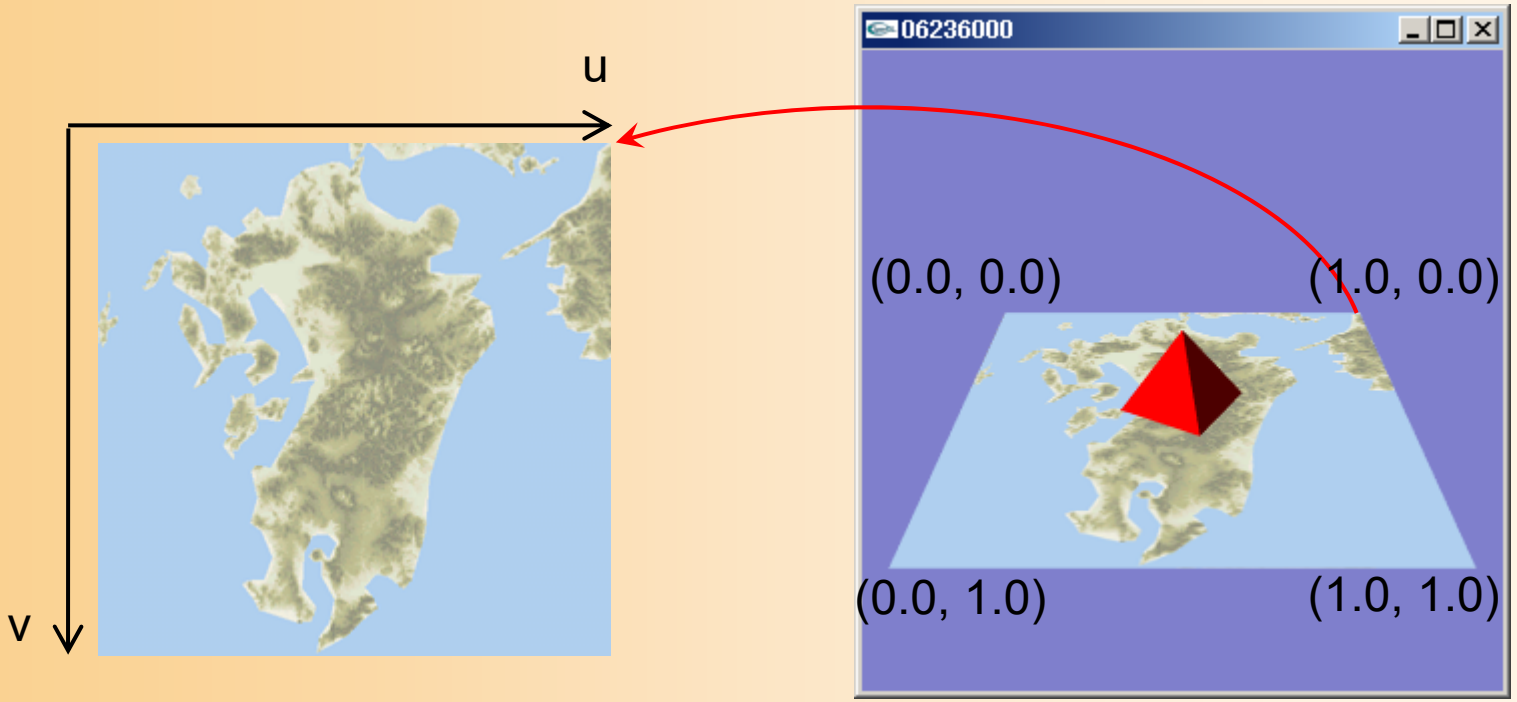
- テクスチャマッピングの適用方法(1)
  - GL_REPLACE
    - 単純にテクスチャ画像を貼り付ける
    - 光源処理は行われないので、常に色は一定
  - GL_DECAL
    - テクスチャ画像のアルファ値に応じて、ポリゴンとテクスチャの色を混合
    - ポリゴンの一部にだけ、テクスチャ画像を貼り付けることができる
    - テクスチャ画像にアルファ値が設定されていなければ GL_REPLACE と同じになる

テクスチャマッピングの適用方法(2)

- テクスチャマッピングの適用方法(2)
  - GL_MODULATE
    - ポリゴンの色にテクスチャ画像の値をかける
    - 光源による明るさの変化を適用できる
      (この場合、通常、ポリゴンの色は、白で描画する)
  - GL_BLEND
    - 別に設定した固定色とポリゴンの色を、テクスチャ画像のアルファ値に応じて混合
    - テクスチャ画像には、画像データの代わりに、混合比の情報を格納しておく

5. テクスチャマッピング

- 地面にテクスチャマッピングを適用して描画
  - 各頂点に、テクスチャ座標(u,v)を指定



```
// 地面を描画（テクスチャマッピング）
glEnable( GL_TEXTURE_2D );
glTexEnvf( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL );

int  i, j;
glBegin( GL_QUADS );
        glNormal3f( 0.0, 1.0, 0.0 );
        glColor3f( 1.0, 1.0, 1.0 );
        for ( i=0; i<10; i++ )
        {
                for ( j=0; j<10; j++ )
                {
                        glTexCoord2f( (i+1) * 0.1, (j+1) * 0.1 );
                        glVertex3f( i - 4.0, 0.0, j - 4.0 );
                        glTexCoord2f( (i+1) * 0.1, j * 0.1 );
                        glVertex3f( i - 4.0, 0.0, j - 5.0 );
                        glTexCoord2f( i * 0.1, j * 0.1 );
                        glVertex3f( i - 5.0, 0.0, j - 5.0 );
                        glTexCoord2f( i * 0.1, (j+1) * 0.1 );
                        glVertex3f( i - 5.0, 0.0, j - 4.0 );
                }
        }
glEnd();

glDisable( GL_TEXTURE_2D );
```

テクスチャ座標の計算(1)

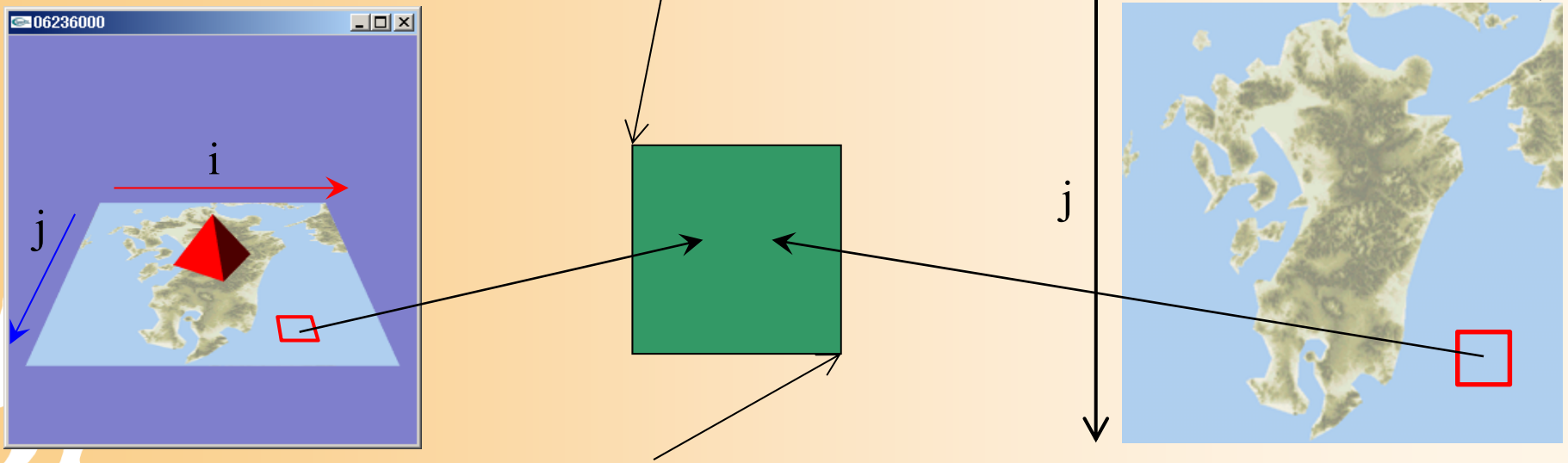
- このプログラムでは、10×10の四角形を並べて地面を描画する
- 各四角形を描画するときに、4つの頂点のテクスチャ座標が適切な値になるように、i, j にもとづいて計算する
  - 例えば、i=0, j=0 のときには、4頂点のテクスチャ座標は(0.1, 0.1)(0.1, 0.0)(0.0, 0.0)(0.0, 0.1)
  - i=9, j=9 のときには、4頂点のテクスチャ座標は(1.0, 1.0)(1.0, 0.9)(0.9, 0.9)(0.9, 1.0)

テクスチャ座標の計算(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）

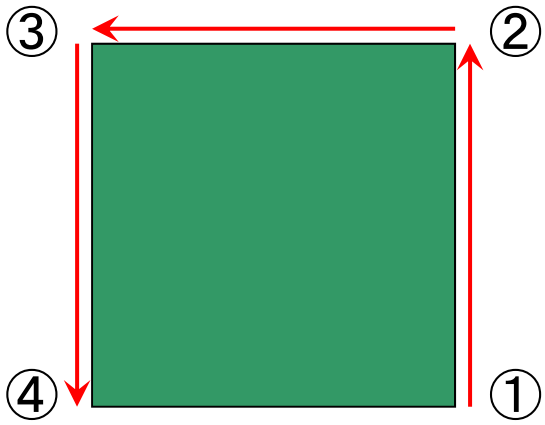
テクスチャ座標（i＊0.1 – 5, j＊0.1 - 5）



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）

```
// 地面を描画
int  i, j;
glBegin( GL_QUADS );
        glNormal3f( 0.0, 1.0, 0.0 );
        glColor3f( 0.5, 0.8, 0.5 );
        for ( i=0; i<10; i++ )
        {
                for ( j=0; j<10; j++ )
                {
                        glTexCoord2f( (i+1) * 0.1, (j+1) * 0.1 );   ①
                        glVertex3f( i - 4.0, 0.0, j - 4.0 );
                        glTexCoord2f( (i+1) * 0.1, j * 0.1 );       ②
                        glVertex3f( i - 4.0, 0.0, j - 5.0 );
                        glTexCoord2f( i * 0.1, j * 0.1 );           ③
                        glVertex3f( i - 5.0, 0.0, j - 5.0 );
                        glTexCoord2f( i * 0.1, (j+1) * 0.1 );       ④
                        glVertex3f( i - 5.0, 0.0, j - 4.0 );
                }
        }
glEnd();
```



複数テクスチャの使用

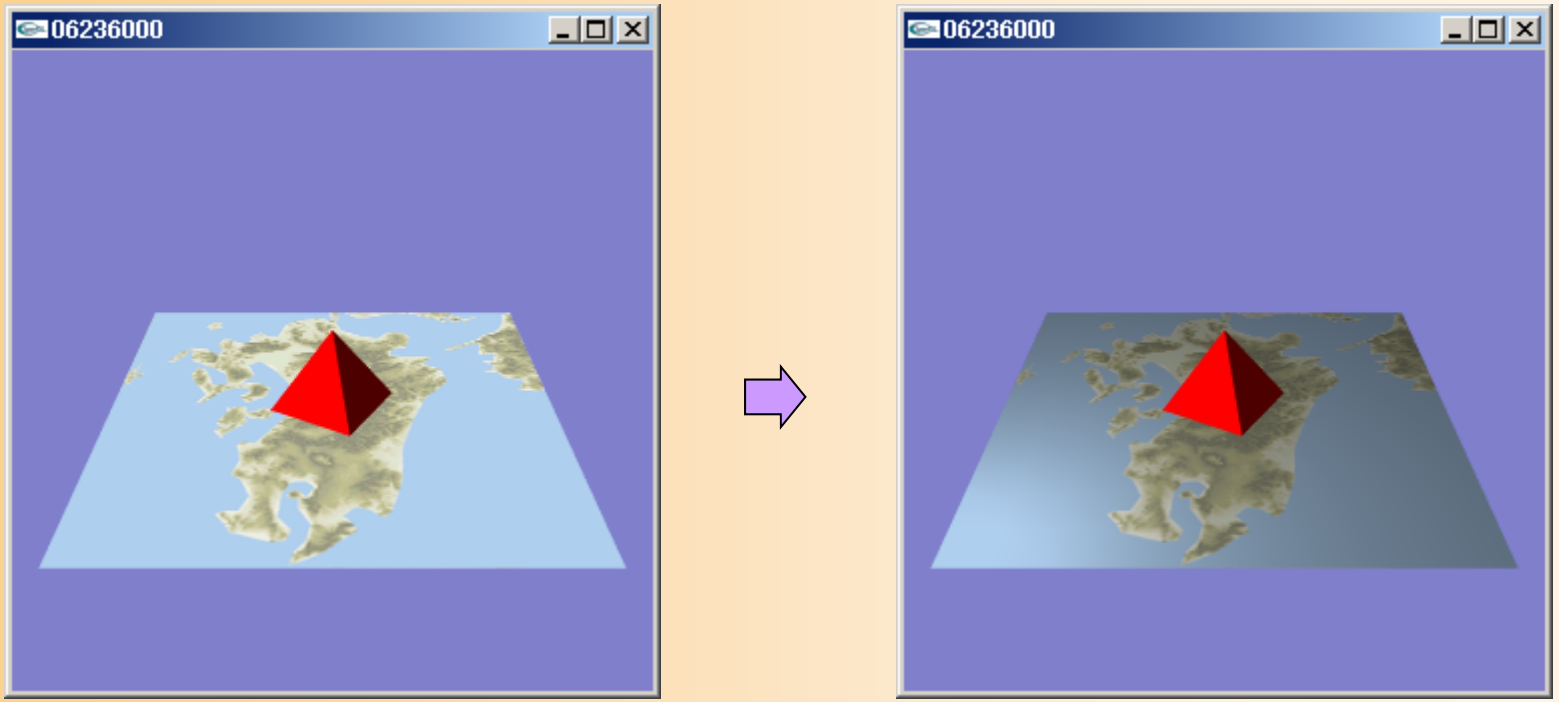
- 複数テクスチャを使用する場合は、テクスチャごとに、テクスチャを切り替える
- 毎回、glTextureImage2D()を行うと時間がかかる
- glTextureBind()関数で、登録したテクスチャ情報を記録しておき、切り替えられる
- テクスチャの切り替えはなるべく少ない方が効率的
  - 同じテクスチャ画像を使用するポリゴンはまとめて描画する
- 本演習では、複数テクスチャの使用は扱わない

テクスチャマッピングの演習

- 以上の処理を記述して、テクスチャマッピングを試してみる
- テクスチャマッピングの適用方法を変更してみる
- テクスチャ座標を変更してみる

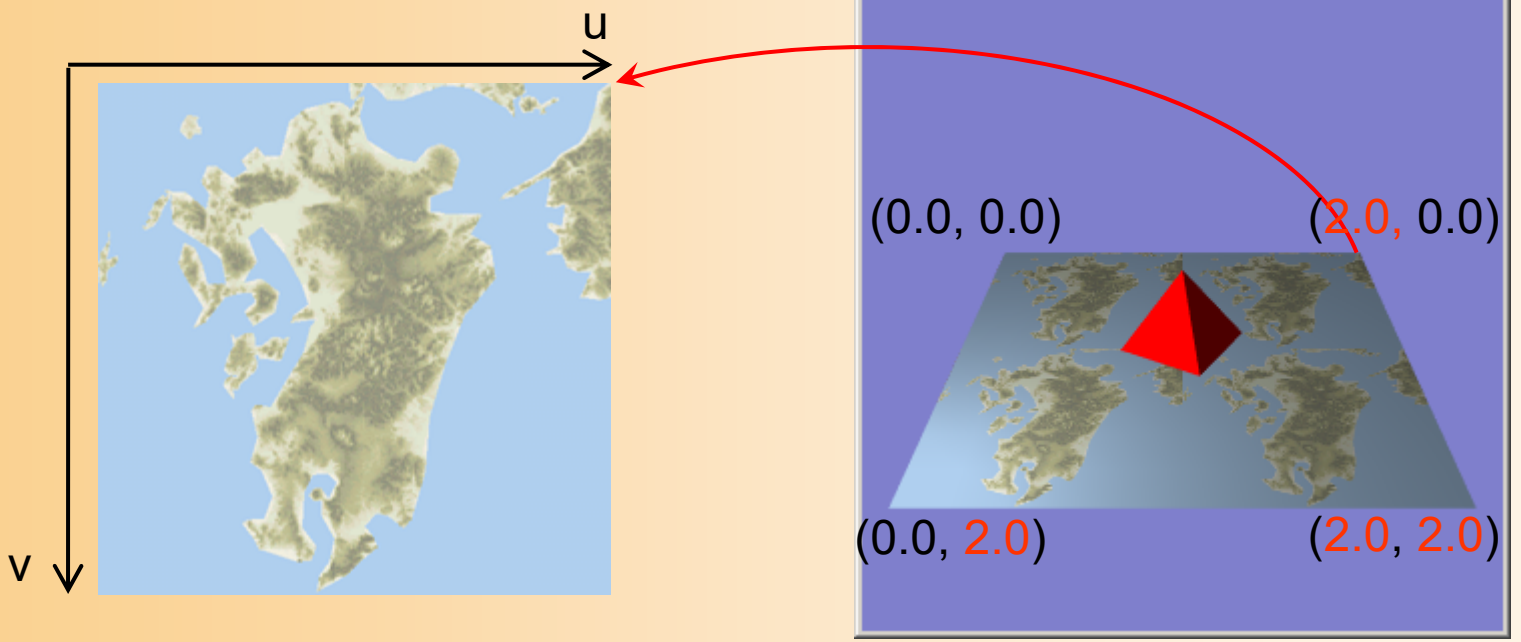
適用方法の変更

- 適用方法を変更してみる
  - GL_DECALから、GL_MODULATEに変更
  - 光源処理の影響を受ける



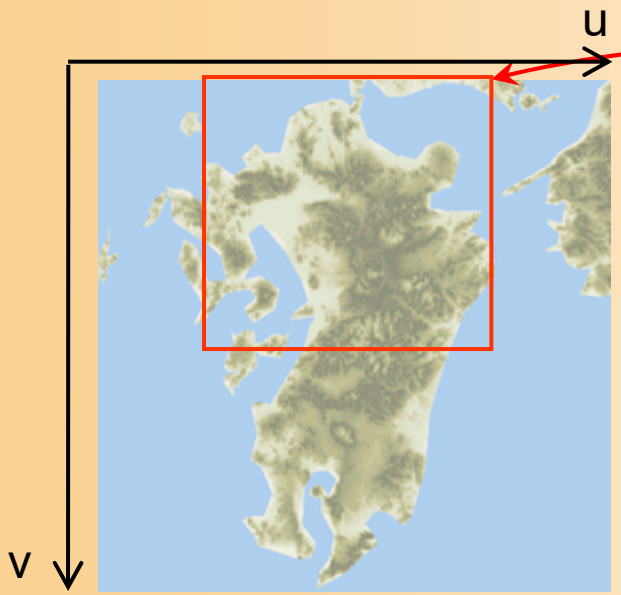
テクスチャ座標の変更

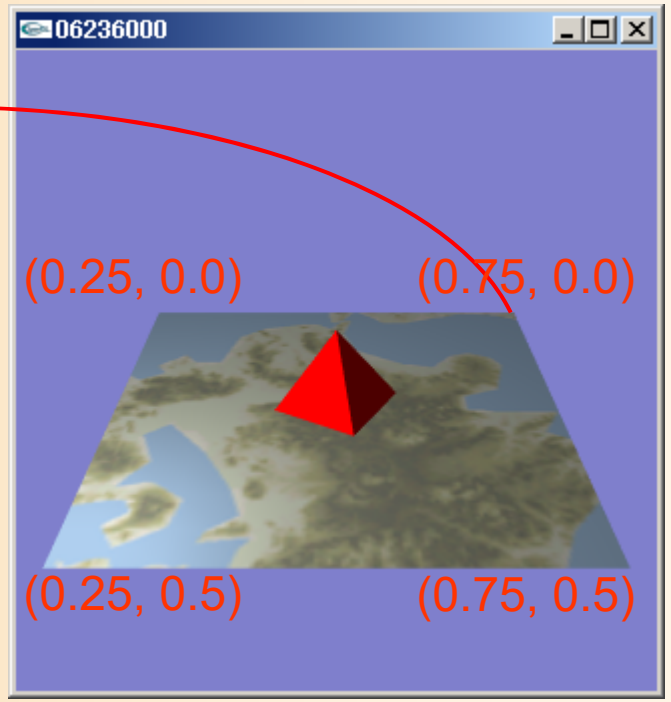
- 各頂点の u,v 座標を2倍にしてみる
  - GL_TEXTURE_WRAP を、GL_REPEATE に設定しているので、0.0~1.0の座標を指定すると繰り返えし貼られる



演習課題(2)

- 下図の通り、テクスチャの一部のみが貼り付けられるように、地面の描画を修正せよ
  - 各頂点の(u,v)座標を適切に修正





演習課題の提出

- 光源位置の設定の変更
- テクスチャマッピング

- 上記の両方の課題を実現した、ひとつのプログラムを、Moodleから提出
  - ファイル名は、学生番号.cpp とする
  - 必ず、両方とも完成したプログラムを提出すること（部分点はなし）
  - 時間内に終わらなければ、締め切りまでに提出
- 締め切り 8月9日（金）18:00（厳守）

まとめ

- 前回・前々回の復習
- シェーディング（光源設定の変更）
- テクスチャマッピング
- これまでの OpenGL演習で学習した内容をもとに、レポート課題を完成させること

次回予告

- アニメーション
- 授業のまとめ

次回予告

- アニメーション
  - 動きのデータを扱う技術