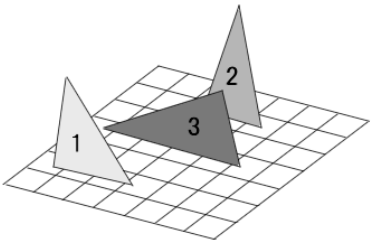
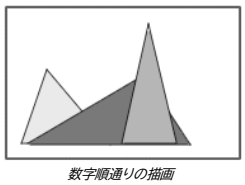
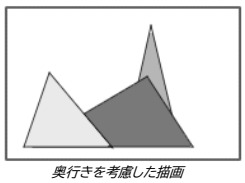
04 – オブジェクト描画

深度テスト

カメラで自由に見回すことが出来るようになると、描画しているオブジェクトの前後関係がおかしいことが分かる。これは、見た目は 3D だが 2D のやり方で描画しているためである。




数字順通りの描画

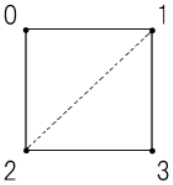

奥行きを考慮した描画

2D の描画は、ノートへのラクガキや写真の切り貼り、シャドーボックスのように、どんどん上に重ねて上書きするように描画する。3D の描画は描き順に関係なく、オブジェクト同士の前後関係が正しい順番になるように描画する。画素(ピクセル)の場所がどのくらいの奥行きを持つかを「　　」というデータで「　　」に保存しておき、新しく描画されるときに保存されいている深度値と、これから描画する深度値を比較して、より手前にあれば描画、そうでなければ描画しないといったことを行う。最終的には一番手前にあるものだけが描画されるようになる。

このように隠れて見えなくなる部分を消去する処理を「　　」という。

```
glEnable(GL_DEPTH_TEST);  // 深度テストの有効化
……
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  // 深度バッファのクリア
```

頂点配列



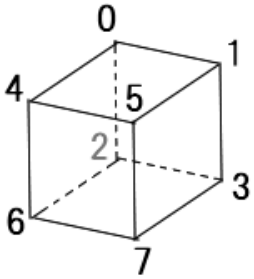
複雑なオブジェクトを描画しようとする場合、オブジェクトの頂点一つ一つを指定して複数の三角形を描画するのは面倒な処理となる(glVertex3f で立方体を描画する…)。また、関数を呼び出す度に頂点を設定し直しているため処理負荷にもなってくる。

そこで、一つずつ渡すのではなく、配列としてまとめて渡すことで上記の問題が解決できる。

```
glEnableClientState(GL_VERTEX_ARRAY); // 頂点配列の有効化
……
float pos[] = {-0.5f,0.5f,  -0.5f,-0.5f,  0.5f,-0.5f}; // 2Dの三角形
glVertexPointer(2, GL_FLOAT, 0, pos); // 座標の数(2D-2/3D-3), 座標のデータ型, 頂点間のデータ数, データ
glDrawArrays(GL_TRIANGLES, 0, 3); // 座標データ順に描画[プリミティブ, データの開始位置, 処理するデータ数
```

2D の三角形を利用することはあまりない。3D 空間上で、XZ 平面に広がるポリゴンを描画してみよう。

インデックス



上記のやり方では、配列に格納されている座標の順番でしか三角形が描画されない。複雑なオブジェクトを描画する場合、複数の頂点を何回も指定しなければならない。そうなると、頂点データに重複した無駄なデータが増えてメモリの効率が悪くなったり、処理する頂点数が多くなって負荷がかかってしまったりと配列にした意味が薄れてしまう。

そこで、三角形ごとに使用したい頂点を番号(インデックス)で指定することで、重複した無駄なデータを生まず複雑なオブジェクトを描画することが出来る。

```
glVertexPointer(3, GL_FLOAT, 0, /*Boxの座標*/);
glDrawElements(GL_TRIANGLES, /*Box のインデックス数*/, GL_UNSIGNED_BYTE, /*Box のインデックス*/);
```

基本的な図形はデバッグに利用される。ボックスだけでなく、球(スフィア)の描画も挑戦してみよう！