AGL Programming Guide

このドキュメントでは、Mac OS Xに実装されているAGLルーチンの使い方について解説します。AGLを用いれば、Carbon環境においてOpneGL APIを実行してウィンドウ上に3Dオブジェクトを描画することが可能です。AGLは、OpenGLとCarbonアプリケーションの仲を取りもつ階層だと考えてください。

（1）AGLのコンセプト

AGLの仕組みを理解するためには、幾つかの用語やコンセプトを理解しておく必要があります。例えば、ピクセルフォーマット（Pixel Formats）、仮想スクリーン（Virtual Screens）、レンダリングコンテキスト（Rendering Contexts）、描画対象オブジェクト（Drawable Objects）、ピクセルバッファ（Pixel Buffers）などです。まずは、こうした用語について順次解説していきます。AGL APIの先頭はすべてaglと表記され、その引数や処理に用いられる各パラメータについてはagl.hで参照可能です。

Carbon環境でOpenGLによる描画を行うには、AGLルーチンを使い、ピクセルフォーマットを選択し、レンダリングを実行するためのコンテキストを作成します。そして最後に描画対象となるオブジェクトを作成し、適切なOpenGLコマンドを実行した後に描画先メモリバッファをスワップ（切り換え）することで、ウィンドウ上に3Dオブジェクトが描画されます。こうした手順については、「AGLタスク」の章で詳しく解説します。

・ピクセルフォーマット（Pixel Formats）

OpenGLにおけるピクセルフォーマットとは、描画先メモリバッファのカラーフォーマットのことを言います。例えば、RGBα各コンポーネントの順番や階調などです。加えて隠面処理のためのデプス値やステンシル（クリッピング用2値バッファ）などの情報も含まれています。AGLでピクセルフォーマットを参照する場合には、AGLPixelFormatデータタイプが用いられます。このデータタイプは隠蔽（Opaque）されており、内部構造を知ることはできません。AGLはMac OS Xのグラフィックデバイスに関連するピクセルフォーマットの一覧を管理しています。aglChoosePixelFormat()に適切なアトリビュートの配列を渡すことで、最初のピクセルフォーマット（AGLPixelFormat）を得ることができます。もし次のピクセルフォーマットが必要な場合にはaglNextPixelFormat()を用いることが可能です。

aglChoosePixelFormat()に渡すアトリビュートは、レンダリング関連（描画機能を示す）とサーフェス関連（描画対象を示す）の2つのカテゴリに分かれます。例えば、AGL_ALL_RENDERERS, AGL_SINGLE_RENDERER,AGL_ACCELERATEDといったパラメータはレンダリング関係のアトリビュートであり、AGL_PIXEL_SIZE, AGL_RGBA, AGL_STEREO, and AGL_RED_SIZEなどはサーフェス関係です。

ピクセルフォーマット用アトリビュート配列は、以下のようにaglNextPixelFormat()に渡すことでAGLPixelFormatを得ます。AGL_NONEはアトリビュート配列の最後（ターミネータ）を意味しています。

```
GLint  atb[]={ AGL_RGBA,AGL_DEPTH_SIZE,16,AGL_NONE };
AGLPixelFormat   fmt;

fmt=aglChoosePixelFormat( NULL,0,atb );
```

・仮想スクリーン（Virtual Screens）

仮想スクリーンはOpenGLが画像描画を行うためのハードウェア、レンダラー、ピクセルフォーマットのコンビネーションです。仮想スクリーンを切り替えれば、通常レンダラーも切り替わります。OpenGLは、アトリビュートを指示することで作成されたピクセルフォーマットにより仮想スクリーンのリストを生成します。AGLはピクセルフォーマットリストの範囲内で仮想スクリーンにユニークな番号を与えます。

Mac OS X環境で、1枚のビデオカードでマルチスクリーンを利用しており（Dual-Head）、その2つのスクリーンをまとめてひとつのスクリーンとして扱う場合を考えてください。片方のスクリーンから他方のスクリーンへウィンドウを移動しても、OpenGLによる描画が正しく行われるのは、両方をまとめてひとつの仮想スクリーンとして管理しているためです。もしシングルディスプレイを利用していれば、リスト内の仮想スクリーンはひとつだけしか存在しません。

ピクセルフォーマットの仮想スクリーンリストは、共有されているレンダリングコンテキストの能力を決定します。コンテキストを共有する場合には、それに同じ仮想スクリーンリストが割り当てられていなければいけません。同じピクセルフォーマットでなくてもよいですが、同じ仮想スクリーン（レンダラー）リストが必要であることに注意してください。これを実現するためには、コンテキストに対して同じピクセルフォーマットを用意するか、特定のひとつのレンダラーを所有する別のピクセルフォーマットを用意します。

・描画対象オブジェクト（Drawable Objects）

Mac OS Xでの描画対象オブジェクトはOpenGLシステム外に割り当てられますが、それをOpenGLのフレームバッファとして用いることも可能です。描画対象オブジェクトの種類としては、ウィンドウ、ビュー（View）、ピクセルバッファ、オフスクリーンメモリ領域、フルスクリーン・グラフィックデバイス等があります。ウィンドウはCarbon環境のみで、ビューはCocoa環境のみで、描画対象オブジェクトとして割り当て可能です。それ以外は、CarbonとCocoa両環境で割り当てることが可能です。

以下のように、AGLではaglSetDrawable()を使い、Carbonウィンドウを描画対象オブジェクトとして割り当てます。割り当てている描画対象オブジェクトのデータタイプはAGLDrawableですが、これはCGrafPtrと一致しています。

```
WindowRef     window;  // 収得済みとする
AGLContext    ctx;          // 収得済みとする

aglSetDrawable( ctx,GetWindowPort( window ) );
```

描画対象オブジェクトは、必ずAGLレンダリングコンテキストに関連ずけられています。ウィンドウ以外の割り当てには、フルスクリーンであればaglSetFullScreen()を、オフスクリーンメモリ領域であればaglSetOffScreen()を利用します。

・レンダリングコンテキスト（Rendering Contexts）

AGLレンダリングコンテキストはOpenGLに関する各種状態を保持しているオブジェクトで、そのデータタイプはAGLContextです。このデータは隠蔽（Opaque）されており内部構造は未知です。AGLレンダリングコンテキストは、aglContextCreate()に対して取得済みのピクセルフォーマット（AGLPixelFormat）を渡すことで作成します。その後、描画対象オブジェクトを設定し、aglSetCurrentContext()を実行することで、OpenGLコマンドの実行結果がそのコンテキストに反映されるようになります。つまり、指定された描画対象オブジェクトに対して、OpenGLコマンドによる2Dや3D描画が可能になるわけです。

```
WindowRef       window;  // 収得済みとする
AGLPixelFormat fmt;       // 収得済みとする
AGLContext      ctx;

if( ctx=aglCreateContext( fmt,NULL );
```

```
if( aglSetDrawable( ctx,GetWindowPort( window ) )==GL_TRUE  )
    aglSetCurrentContext( ctx  );
```

AGLレンダリングコンテキストの作成時に、NULLの代わりに別のコンテキストを代入することで、OpenGLオブジェクトを共有することが可能です。OpenGLオブジェクトには、テクスチャ、プログラムとシェーダのリスト、バーテックスアレイ、バーテックスバッファ、ピクセルバッファ、フレームバッファなどが含まれます。また、特定のレンダリングコンテキスとに対してOpenGLコマンドはリエントラントではありません。よって、同期処理をぜず、2つのスレッドから同じコンテキストをターゲットにしたコマンドを発生する時には注意が必要です。こうした同期処理はOpenGLシステム外で行い、CGLで用意されているロックやアンロックコマンドを用いるとよいでしょう（CGL Referencesを参照）。

・ピクセルバッファ（Pixel Buffers）

ピクセルバッファは描画対象オブジェクトのひとつです。ピクセルバッファはOpenGLのテクスチャバッファとして利用することが可能なオフスクリーンメモリ領域です。この仕組みは、Mac OS X 10.3から導入されました。

ピクセルバッファはaglCreatePBuffer()で作成してから、aglSetPBuffer()で描画対象オブジェクトとしてレンダリングコンテキストに割り当てます。aglTexImagePBuffer()は、OpenGLのglTexImage2Dコマンドと同様の働きをし、3Dや2D描画プリミティブにテクスチャを貼り付けます。

・AGLはどのように働くのか？

AGLはアプリケーション起動時にすべてのOpenGL Plug-inレンダラーを登録します。そして、アプリケーションがピクセルフォーマットを指定すると、それに最適なレンダラーを選択し、OpenGLコマンドをそのレンダラーに送ります。ユーザがウィンドウを切り替えた時にピクセルフォーマットが変われば、それに最適な新しいレンダラーを自動で割り当てます。このようにアプリケーション側では、OpenGLのレンダラーの挙動や管理を気にする必要はありません。

また、AGLはマルチモニタ環境をまたいでレンダラーを管理します。この時、2つのモニターのカラー階調などの性能が異なったとしても、両モニタ間をウィンドウを移動させることでOpenGLによる描画に影響が出ることはありません。もし、AGLによるレンダラーの選択をより効果的に制御したい場合には、現状に最適なレンダラーを番号（Renderer ID）指定で呼び出すことも可能です。

アプリケーションがマルチモニタ環境でピクセルフォーマットを選ぶ時、AGLは各モニタに最適（最大パフォーマンス）なレンダラーを見つけようとします。この場合、AGLは、同一のレンダラーで大丈夫なのか、別のレンダラーを利用するのかを、描画パフォーマンスが最大になるように考慮して適切に判断します。

最後に、今まで解説してきた処理をまとめたcreateOpenGLContext()ルーチンを紹介しておきます。このルーチンは、OpebnGLの描画対象とするウィンドウ（WindowRef）を渡し、作成されたAGLレンダリングコンテキスト（AGLContext）を返します。まずは、アトリビュートリストから作成したピクセルフォーマットを用いてAGLレンダリングコンテキストを得ます。その後、ウィンドウのCGrafPtrを描画対象オブジェクトに設定してから、そのAGLレンダリングコンテキストをカレントにしています。

```
short createOpenGLContext( WindowPtr window,AGLContext *ctx )
{
    GLint               atb[]={ AGL_RGBA,AGL_DEPTH_SIZE,16,AGL_NONE };
    short               ret=1;                          // アトリビュートリスト
    AGLPixelFormat      fmt;

    if( fmt=aglChoosePixelFormat( NULL,0,atb ) ) // ピクセルフォーマットの選択
    {
        if( *ctx=aglCreateContext( fmt,NULL ) )      // 新規AGLコンテキストを作成
        {
            if( aglSetDrawable( *ctx,GetWindowPort( window ) )==GL_TRUE )
            {                           // ウィンドウを描画対象オブジェクトとする
                if( aglSetCurrentContext( *ctx  )==GL_TRUE )
                    ret=noErr;      // カレントコンテキストに設定
            }
            else
                aglDestroyContext( *ctx  ); // AGLコンテキストの破棄
        }
        aglDestroyPixelFormat( fmt );    // ピクセルフォーマットの破棄
    }
    return( ret );
}
```

（2）AGLのタスク

この章では、AGL APIを利用した一般的なプログラミングを紹介します。また、AGLを用いたプログラミングのポイントやガイドラインも紹介しています。

・ピクセルフォーマットのセットアップ

ピクセルフォーマットのセットアップに関しては以下のような要点が上げられます。

ハードウエァアクセラレーションされたレンダラーのみを選択する場合には、アトリビュートとしてAGL_ACCELERATEDとAGL_NO_RECOVERYの両方を設定します。

ソフトウェアによるレンダラーを選択する場合には、AGL_RENDERER_IDアトリビュートを設定し、そのパラメータとしてAGL_RENDERER_GENERIC_IDを指示します。浮動小数点レンダラーを用いる場合には、代わりにAGL_RENDERER_GENERIC_FLOAT_IDを指示します。

アトリビュートにAGL_OFFSCREENを用い、システムのメモ領域にレンダリングを実行する場合には、ハードウエァアクセラレーションは利きません。もし利かせたい場合には、後述するピクセルバッファを代わりに利用します。

Mac OS Xの場合、ウィンドウサーバ自身がすでにダブルバッファによる描画を利用していますので、AGL_DOUBLEBUFFERを指定しなくともフリッカーレスの描画を実現することが可能です。この場合、描画開始の指示にはaglSwapBuffers()ではなくglFlush()を用います。

・ディフォルト以外のレンダラーの使用

もし、仮想スクリーンを切り換えることで利用しているディフォルトレンダラーを変更したい場合には、以下のルーチンを利用します。すると、その仮想スクリーンを所有しているピクセルフォーマットにマッチしたレンダラーが使用されます。

aglSetVirtualScreen(ctx, screen);

ctx　　対象となるAGLレンダリングコンテキスト
screen　対象となる仮想スクリーンID番号

・レンダラーの列挙

AGLピクセルフォーマットに関連づけられたレンダラーの個数を調べるのには、以下のような処理を実行します。

```
int i = 0;

while (pix)
{
    pix = aglNextPixelFormat (pix);      // 次のピクセルフォーマットを得る
    i++;                                 // カウンターを増やす
}
```

現在のピクセルフォーマットで用いられているレンダラーID番号を得るには、以下のような処理を実行します。

```
curr_virtual_screen = aglGetVirtualScreen (ctx);  //   現在の仮想スクリーンを得る
aglDescribePixelFormat (pix, AGL_VIRTUAL_SCREEN, &format_virtual_screen);
                        // ピクセルフォーマットとのための仮想スクリーンを得る
while (format_virtual_screen != curr_virtual_screen && pix)
{                       //  ピクセルフォーマットの仮想スクリーンが一致するまで繰り返す
     pix = aglNextPixelFormat (pix);
     aglDescribePixelFormat (pix, AGL_VIRTUAL_SCREEN,
                                    &format_virtual_screen);
                        // ピクセルフォーマットとのための仮想スクリーンを得る
}
if (pix)                // 仮想スクリーンが一致した場合
   aglDescribePixelFormat(pix, AGL_RENDERER_ID, &renderer);
                        // 仮想スクリーンのレンダラーを得る
```

・ディスプレイの書き換え（Vertical Retrace）との同期

レンダリングをディスプレイの書き換え周波数（Vertical Retrace）と同期させるには、アトリビュートとしてAGL_SWAP_INTERVALを指定し、aglSetInteger()で適切な周波数を設定します。OpenGLはディフォルトでは同期していません。以下の処理でディスプレイの書き換え周波数と同期を取ることが可能です（外部モニターとの同期は不可）。

```
GLint sync = 1;
aglSetInteger (ctx, AGL_SWAP_INTERVAL, &sync);  // syncに1を代入する
```

・コンテキストの共有

AGLレンダリングコンテキストを作成する時、他のレンダリングコンテキストとOpenGLオブジェクトを共有することが可能です。OpenGLオブジェクトには、テクスチャ、プログラムとシェーダのリスト、バーテックスアレイ、バーテックスバッファ、ピクセルバッファ、フレームバッファなどが含まれます。これ以外のコンテキストの状態は共有されません。共有されたオブジェクトは明示的に破棄されるか、最後のコンテキストが破棄されるまで保持されます。

AGLレンダリングコンテキストの共有には制限が付きます。コンテキストの共有を可能にしたい場合には、以下の2つの手法のうちどちらかを取ります。

（１）共有したいすべてのコンテキストに完全に同じピクセルフォーマットを用意する。

```
#include <AGL/agl.h>

GLint attrib[] = {AGL_RGBA, AGL_DOUBLEBUFFER, AGL_NONE};

AGLPixelFormat aglPixFmt = aglChoosePixelFormat (NULL, 0, attrib);
AGLContext aglContext1 = aglCreateContext (aglPixFmt, NULL);
AGLContext aglContext2 = aglCreateContext (aglPixFmt, aglContext1);
        //  2つめのコンテキストを最初の物と共有させて作成する
```

（２）特定のひとつのレンダラーをすべてのピクセルフォーマットで選択する。

```
#include <AGL/agl.h>

GLint attrib[] = {AGL_RGBA, GL_DOUBLEBUFFER, AGL_NO_RECOVERY,
                                                    AGL_NONE};
        // ソフトレンダラーを選択しないためAGL_NO_RECOVERYを指定する
GDHandle display = GetMainDevice ();
AGLPixelFormat aglPixFmt = aglChoosePixelFormat (&display, 1, attrib);
```

もし、AGLレンダリングコンテキストを共有した場合には以下の点に注意してください。単一ディスプレイ（仮想スクリーン）をサポートしたピクセルフォーマットを設定した場合、もし描画対象に割り当てたウィンドウが別ディスプレイへ移動すると正常なレンダリングはなされません。これを避けるのにはレンダラーが切り替わるのを見つけて制御する処理を用意する必要があります。また、Mac OS X 10.3より、単独ウィンドウとフルスクリーンコンテキストを共有する能力が追加されました。

・サーフェス（Surfaces）の働き

サーフェス（Surfaces）とは、OpenGLがデータを描画したり読み込んだりする内部バッファを指します。Mac OS Xでは、ウィンドウサーバがこのバッファの内容をウィンドウの内容などと合成してデスクトップに表示しています。AGLでは、この合成方法を制御することが可能です。

（1）サーフェス・バッキングサイズ（Backing Size）の制御

通常、OpenGLの描画用バッファサイズは描画対象オブジェクトと一致しています。例えな、描画対象オブジェクトがウィンドウの時、そのサイズを720X480から1024X768に拡大すると、OpenGLの描画用バッファも同サイズに拡大されます。VRAMの消費量を軽減するなどの理由でこの操作を制御したい場合には、以下の様に、aglSetInteger()を使い、コンテキストオプションのAGL_SURFACE_BACKING_SIZEに適切なバッファサイズを指定した後、AGL_ENABLE_SURFACE_BACKING_SIZEをaglEnable()に渡して制御の開始を指示します。

```
GLint dim[2] = { 720, 480 }
aglSetInteger (ctx, AGL_SURFACE_BACKING_SIZE, dim);
aglEnable (ctx, AGL_ENABLE_SURFACE_BACKING_SIZE);
```

（2）サーフェスの不透明度（Opacity）の設定

通常、OpenGLの描画用バッファとウィンドウン内容との合成では透明度は考慮されていません。これを考慮するように切り換えるには、aglSetInteger()を使いコンテキストオプションのAGL_SURFACE_OPACITYにゼロを設定します。

```
long opaque = 0;
aglSetInteger (ctx, AGL_SURFACE_OPACITY, &opaque);
```

（3）サーフェスの描画順序の設定

通常、OpenGLで描画された内容はウィンドウの一番上に合成されます。この順序を変更するには、aglSetInteger()を使いコンテキストオプションのAGL_SURFACE_ORDERを設定し直します。1を代入すればOpenGLサーフェスが上に、以下のように-1を代入すればウィンドウの内容が上に合成されます。

```
long order = －1;
aglSetInteger (ctx, AGL_SURFACE_ORDER, &order);
```

・ピクセルバッファの使用

ピクセルバッファの機能はMac OS X 10.3以降で利用可能です。使用しているシステム環境でピクセルバッファが利用できるかどうかについては、OpenGLに拡張文字列であるGL_APPLE_pixel_bufferの存在を尋ねます。ピクセルバッファ利用に特殊なハードウェアは必要ありませんが、特定のテクスチャモードはその必要があるかもしれません。現在のシステムにおいて、OpenGLで利用可能な機能を正確に調べたい場合には、以下のURLのテクニカルノート TN2080「Understanding and Detecting OpenGL Availability」を参考にしてください。

http://developer.apple.com/technotes/tn2002/tn2080.html

（a）ガイドライン

ピクセルバッファを有効に活用するには以下のガイドラインに従ってください。

ピクセルフォーマットを作成する時には、アトリビュートとしてAGL_PBUFFERを設定します。この指示でピクセルフォーマットが作成できないとすると、ピクセルバッファの利用は保証されません。

ピクセルバッファのビットデプスやピクセルフォーマットなどは、aglCreatePBuffer()で渡す内部フォーマットからは判断しないでください。それには、ピクセルバッファ用のレンダリングコンテキストに割り当てられているピクセルフォーマットの内容が利用されます。

ピクセルバッファをテクスチャとして利用するためには、その前に必ずaglSetPBuffer()を実行する必要があります。

glTexImage1D()やglTexImage2D()など、通常OpenGLコマンドでテクスチャを利用した時には、aglTexImagePBuffer()を実行する前に、必ずglDeleteTextures()を実行して前のテクスチャを削除しておいてください。

glTexSubImage2D()やglCopyTexImage2D()など、実行することでテクスチャの内容を変えてしまうようなOpenGLコマンドは、ピクセルバッファに対して使用することが許されていません。逆に、glCopyTexImage2D()やglReadPixels()などのOpenGLコマンドでピクセルバッファ内のデータを読み込むことは許されています。

カレントのAGLレンダリングコンテキストを破棄した時には、aglSetCurrentContext()でNULLを設定しておくことが推奨されています。

Mac OS X v10.3.0からv10.3.4までは、描画対象オブジェクトを別の物に切り換える時には、その処理の前に一度aglSetDrawable()にNULLを代入して実行してください。これは、描画対象オブジェクトをピクセルバッファからそうでない物へ切り換える場合についても同じです。Mac OS X 10.3.5からは、そうした処理を行う必要はありません。

（b）ピクセルバッファの作成とセットアップ

ピクセルバッファを作成するには、まず最初にAGL_PBUFFERアトリビュートを設定したピクセルフォーマットを作り、それを割り当てたレンダリングコンテキストを作成します。その後、以下のようにaglCreatePBuffer()でピクセルバッファを作成します。

```
AGLPbuffer     pb;

aglCreatePBuffer (512, 512, GL_TEXTURE_2D, GL_RGBA, 0, &pb );
```

引数には、バッファの幅と高さ、テクスチャのターゲット、カラーフォーマット、ミップマップの最大レベルを渡します。ターゲットとしては、GL_TEXTURE_2D（ノーマル2Dテクスチャ）GL_TEXTURE_RECTANGLE_EXT（一辺が2の階乗サイズではない2Dテクスチャ）GL_TEXTURE_CUBE_MAP（キューブマップテクスチャ）を選択することができます。カラーフォーマットはアルファ値を使うかどうかで、GL_RGBAかGL_RGBを指定します。また、テクスチャのミップマップを使用しない場合には、その最大レベルはゼロと指定します。

ピクセルバッファが作成できたら、aglSetPBuffer()を使い、以下の様にそれを描画対象オブジェクトに指定します。

```
AGLContext    tgContext;   // 取得済み
AGLContext    pbContext;   // 取得済み
GLint              vs;

aglSetCurrentContext( pbContext)
vs=aglGetVirtualScreen (tgContext);
aglSetPBuffer (pbContext, pb, 0, 0, vs);
```

引数は、ピクセルバッファのレンダリングコンテキスト、ピクセルバッファ、フェイス、ミップレベル、テクスチャを利用する仮想スクリーンです。フェイスはテクスチャのターゲットがGL_TEXTURE_CUBE_MAPの場合に限り6種類（種類についてはOpenGLの仕様を参照）から選択し（それ以外はゼロ）、ミップレベルには最大ミップレベル以下の数値を設定します。テクスチャを使う仮想スクリーンのID番号は、aglGetVirtualScreen()にテクスチャを利用するレンダリングコンテキストを渡して得ます。

（c）テクスチャソースとしてピクセルバッファを使用する

aglSetPBuffer()でピクセルバッファを描画対象オブジェクトに指定すると、その後のOpenGLコマンドによる描画はピクセルバッファに対して実行されます。これにより、ピクセルバッファにテクスチャが形成されます。ピクセルバッファに保存されたテクスチャは、以下のaglTexImagePBuffer()を用いてウィンドウに表示される3Dプリミティブなどに貼り付けることができます。aglTexImagePBuffer()に渡すレンダリングコンテキストは、ピクセルバッファの物ではなく、描画対象となるウィンドウなどに割り当てられた物です。この作業は、OpenGLコマンドを用いたテクスチャの貼り付け（Bind）作業とほぼ同じです。つまり、aglTexImagePBuffer()はOpenGLのglTexImage2D()コマンドと同等な働きをするわけです。

```
AGLContext    tgContext;   // 取得済み
AGLPbuffer    pb;               // 取得済み

aglTexImagePBuffer (tgContext, pb, GL_FRONT);
```

ピクセルバッファの使用が終わったら、aglDestroyPBuffer()によりそれを破棄します。重要なことは、ピクセルバッファを破棄する前にはOpenGLのテクスチャオブジェクトを削除しておくことです。また、ピクセルバッファ用に作成したレンダリングコンテキストやピクセルフォーマットは、その他の状況と同様に破棄することができます。

（d）ピクセルバッファを利用する場合の手順

ピクセルバッファを作成し利用するための手順は以下のようになります。ピクセルバッファは、それ自身のピクセルフォーマットとレンダリングコンテキストを持ちます。

（1）ピクセルバッファを描画対象オブジェクトとして設定する。

AGL_PBUFFERアトリビュートを設定し、aglChoosePixelFormat()でピクセルフォーマットを作成し、それをaglCreateContext()に渡してAGLレンダリングコンテキストを得ます。続いてaglCreatePBuffer()でピクセルバッファ本体を作成し、aglSetPBuffer()により描画対象オブジェクトに指定します。

（2）ピクセルバッファへ描画します。

aglSetCurrentContext()でピクセルバッファのコンテキストをカレントにしてから、OpenGLコマンドを用いて（glBegin()やglEnd()）プリミティブの描画を実行します。最後にglFlush()かaglSwapBuffers()を実行すれば、ピクセルバッファ上にテクスチャデータ（OpenGLによる描画結果）が形成されます。

（3）ウィンドウなどのレンダリングコンテキストでテクスチャを利用します。

aglSetCurrentContext()により、AGLレンダリングコンテキストをテクスチャを表示するウィンドウに切り替えます。後は、OpenGにおける通常のテクスチャ処理を実行しますが、この時glTexImage2D()コマンドの代わりにaglTexImagePBuffer()を用います。

```
AGLContext    tgContext;   // 取得済み
AGLPbuffer    pb;          // 取得済み
GLuint        txt;         // テクスチャオブジェクト

glGenTextures (1, &txt);
                  // テクスチャオブジェクトを作成する（OpenGLの仕様を参照）
glBindTexture (GL_TEXTURE_2D, txt);
                  // テクスチャオブジェクトをバインドする（OpenGLの仕様を参照）
glTexParameterf (GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
                  // テクスチャオブジェクトのパラメータを設定（OpenGLの仕様を参照）
aglTexImagePBuffer (txt, pb, GL_FRONT);
                  // ピクセルバッファのテクスチャを貼り付ける
```

（4）ピクセルバッファのテクスチャを描画します。

実際のテクスチャ描画の前には、glEnable()を用いてカレントコンテキストでテクスチャを利用することを許可しておく必要があります。

glEnable(GL_TEXTURE_2D);

その後、glBindTexture()、glBegin()、glEnd()によりOpenGLにおける通常の描画処理を実行します。

（5）不必要になったオブジェクトを破棄します。

aglSetCurrentContext()によりカレントコンテキストを描画ターゲットへと切り換え、glDeleteTextures()コマンドでテクスチャオブジェクトを破棄します。その後、必要であればaglDestroyPBuffer()でピクセルバッファ自身を、aglDestroyContext()でピクセルバッファに割り付けられたレンダリングコンテキストを、aglDestroyPixelFormat()でピクセルフォーマットを順序通りに破棄します。

・AGLマクロを用いたパフォーマンスの向上

通常AGLでは、OpenGLコマンドが発生される度にグローバルのレンダリングコンテキストとレンダラーを参照します。この参照が頻繁に行われると処理のパフォーマンスに影響が出ます。Mac OS Xでは、このオーバーヘッドを解消しパフォーマンスを向上させるために、2つのテクニックが利用できるようになっています。ひとつは、ローカルコンテキストを供給するためのAGLマクロの活用、もうひとつは現在のレンダラーをキャッシュするAGLマクロの活用です。

AGLマクロを利用するには、aglMacro.hヘダーファイルをインクルード（#include）します。このヘッダファイル内には、すべてのOpenGLコマンドがマクロとして定義されています。このマクロを利用することでコンテキストとレンダラーの参照をプリコンパイルすることができます。このうち、レンダリングコンテキストの参照にAGLマクロを活用する方法は2種類あります。

最初の方法は、AGL初期化は通常通り行い、その後OpenGLコマンドを実行する直前にAGL_MACRO_DECLARE_VARIABLES()を実行する方法です。もうひとつの方法は、ローカル変数としてAGLContextであるagl_ctxを定義し、そこに対象コンテキストを代入してからOpenGLコマンドを実行する方法です（詳しくはaglMacro.hを参照）。

次はレンダラーの参照にAGLマクロを活用する方法です。aglMacro.hをインクルードした後に、AGL_MACRO_CACHE_RENDERERを定義（#define）します。そして、OpenGLコマンドを実行する直前に、AGL_MACRO_DECLARE_VARIABLES()に引き続き、AGL_MACRO_DECLARE_RENDERER()を実行します。

（注意）aglMacro.hヘファイルとはAGL Framework内に保存されていますが、現状ではaglContext.hと一緒にプロジェクト内にコピーして用いないと問題があるようです。

本ドキュメントの履歴

オリジナル2005年10月4日　要約2005年12月15日 v1.00