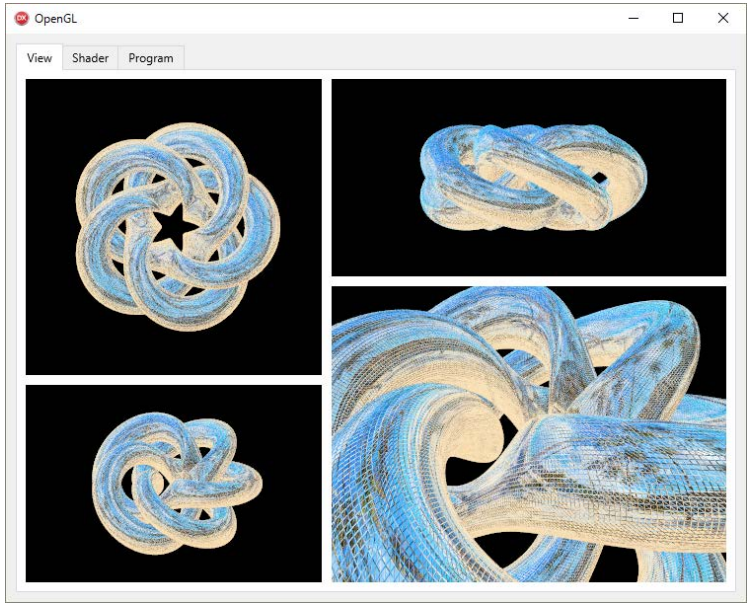
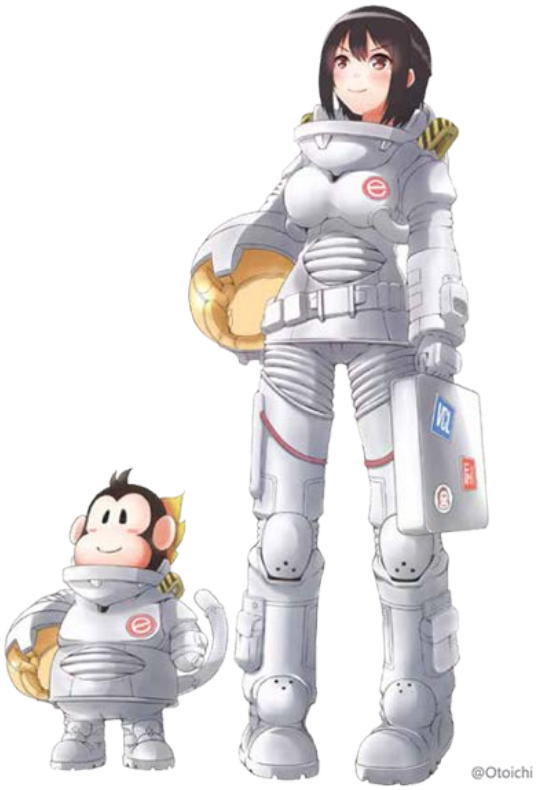
Delphiで超高速OpenGL 2D/3D描画

～ FMX / VCL コンポーネントで驚きの性能実現 ～

第34回 エンバカデロ・デベロッパーキャンプ





慶應義塾大学藤代研究室 特別研究員
＋玉泉山安国院 住職

中山 雅紀

contact@luxidea.net

embarcadero®
DEVELOPER CAMP

アジェンダ

- OpenGLとは？
- OpenGLの初期化
  - FMX の場合
  - VCL の場合
- コマンドの実行
- シーンの構築
- 物体の定義
- OpenGL の歴史
- ミニマムライブラリ
- マキシマムライブラリ

・ＦＭＸ版：github.com/LUXOPHIA/OpenGL
・ＶＣＬ版：github.com/LUXOPHIA/OpenGL_VCL

README.md

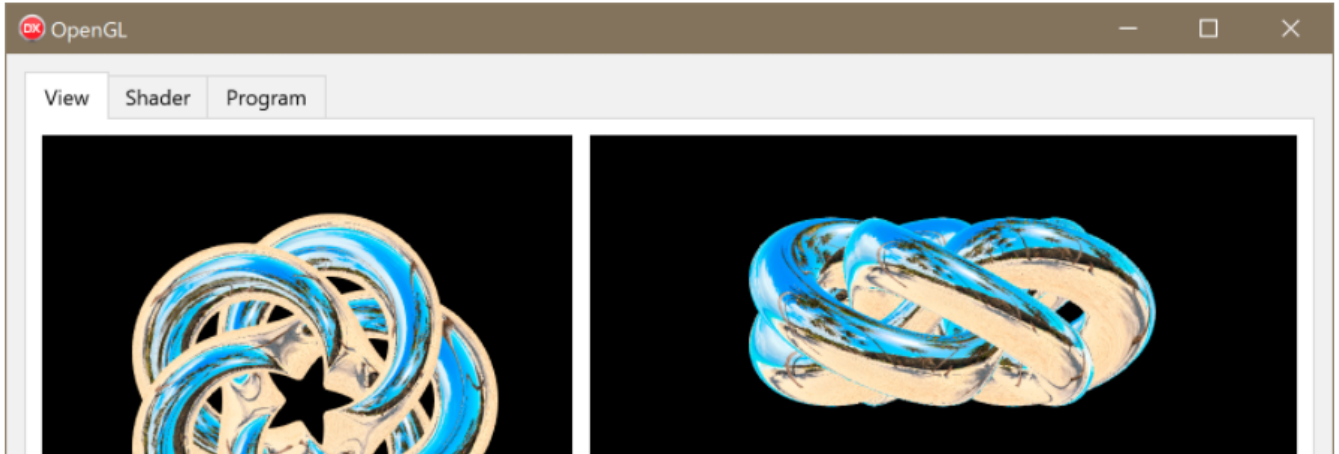
歴史に学ぶ OpenGL

OpenGL のバージョンを辿りながら実装していくことで、新しい API が追加された意図を理解しながら、パイプラインの構造を把握していきます。

- OpenGL 1.0
- OpenGL 1.1
- OpenGL 1.5
- OpenGL 2.1
- OpenGL 3.0

OpenGL

FMX：FireMonkey フレームワークのコンポーネントとして OpenGL の描画領域を埋め込む方法。



自己紹介.身分

- 慶應義塾大学 藤代研究室
  www.fj.ics.keio.ac.jp
  - リサーチフェロー
    - nakayama@fj.ics.keio.ac.jp
    - ＣＧの研究
- 和洋女子大学 山本研究室
  www.a-cad.net
  - 共同研究員
    - ３ＤアパレルＣＡＤの研究
- 玉泉山 安国院
  www.ankokuin.or.jp
  - 住職
    - contact@ankokuin.or.jp
    - 中道思想の研究

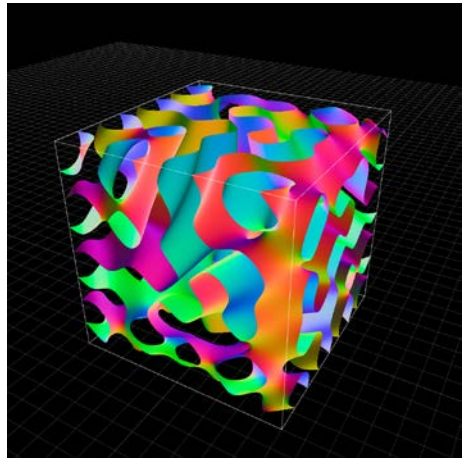

慶應義塾大学 矢上キャンパス

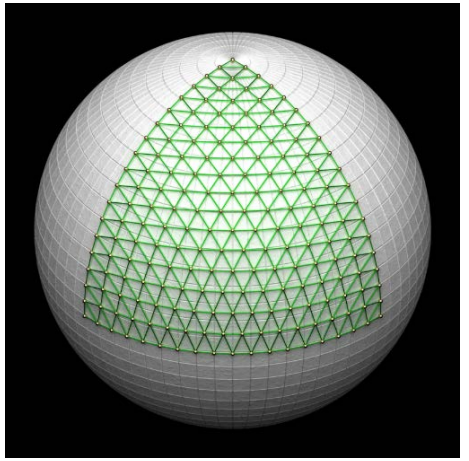
和洋女子大学

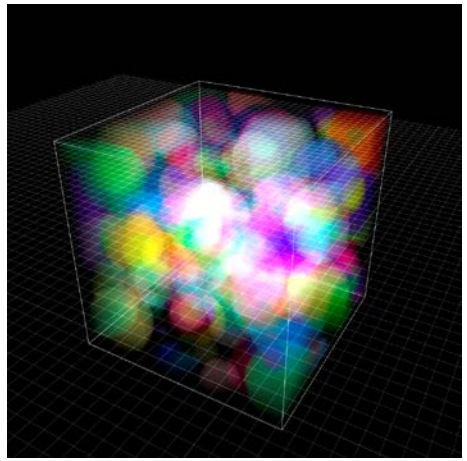

玉泉山 安国院

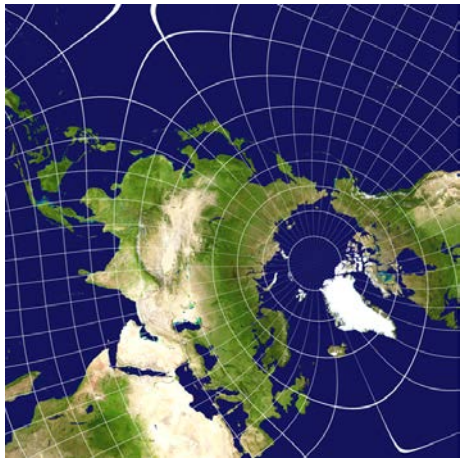
自己紹介.専門

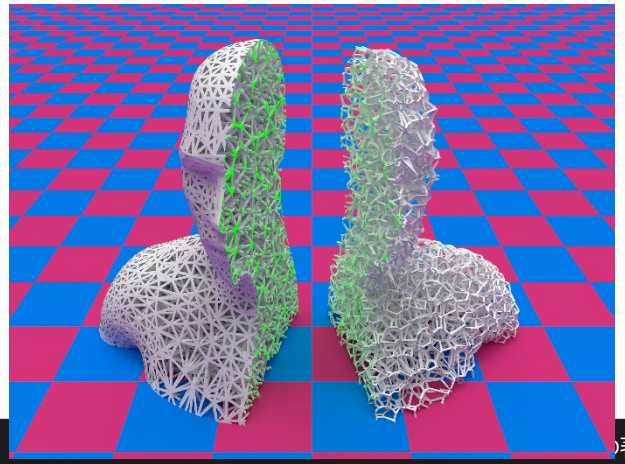
- 球面幾何学
- 写実的レンダリング
- 立体視
- アパレル設計
- 数学可視化
- 体積ベースモデリング
- ３Ｄプリンタ

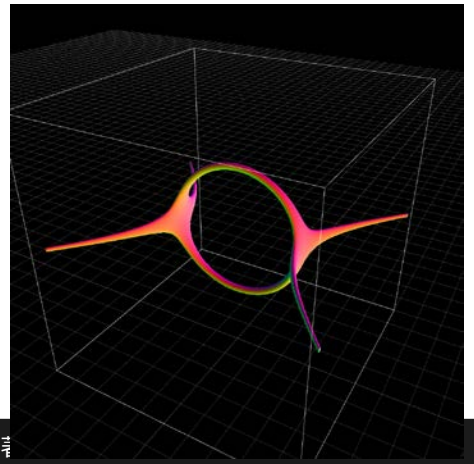


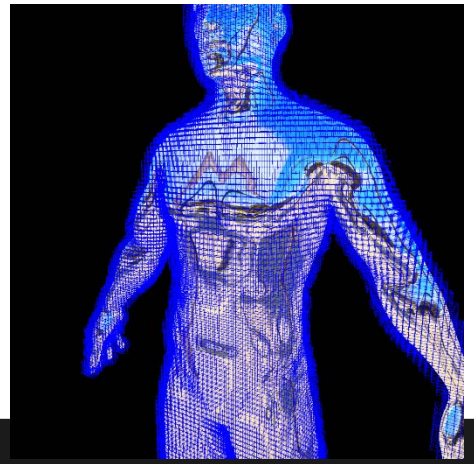


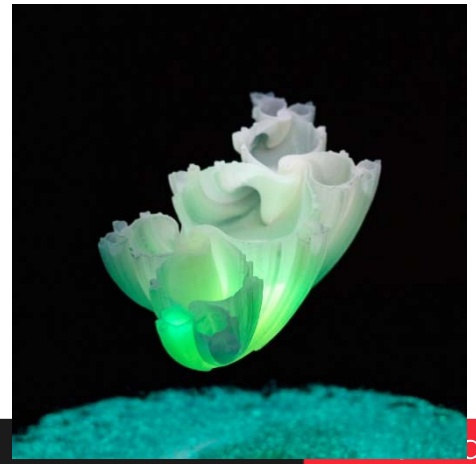








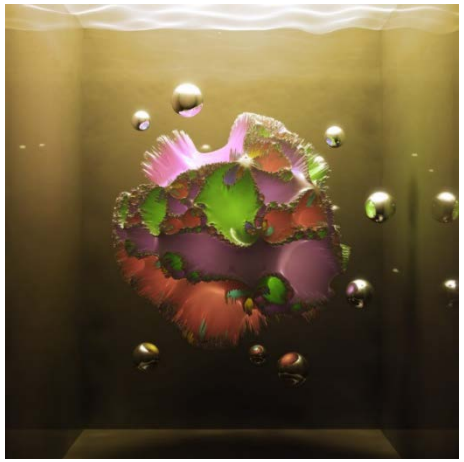


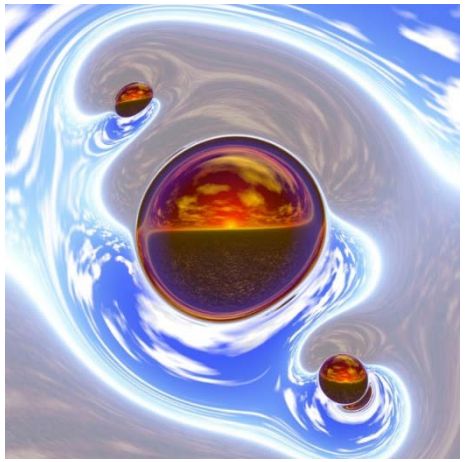


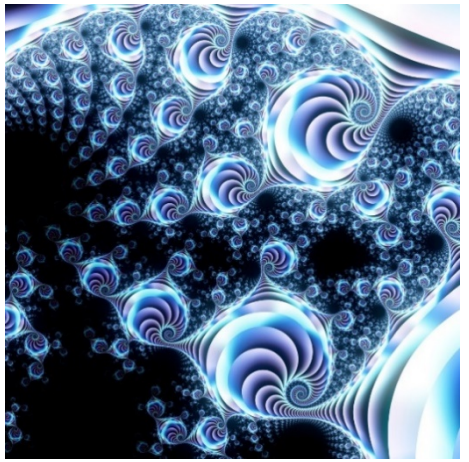
自己紹介.趣味

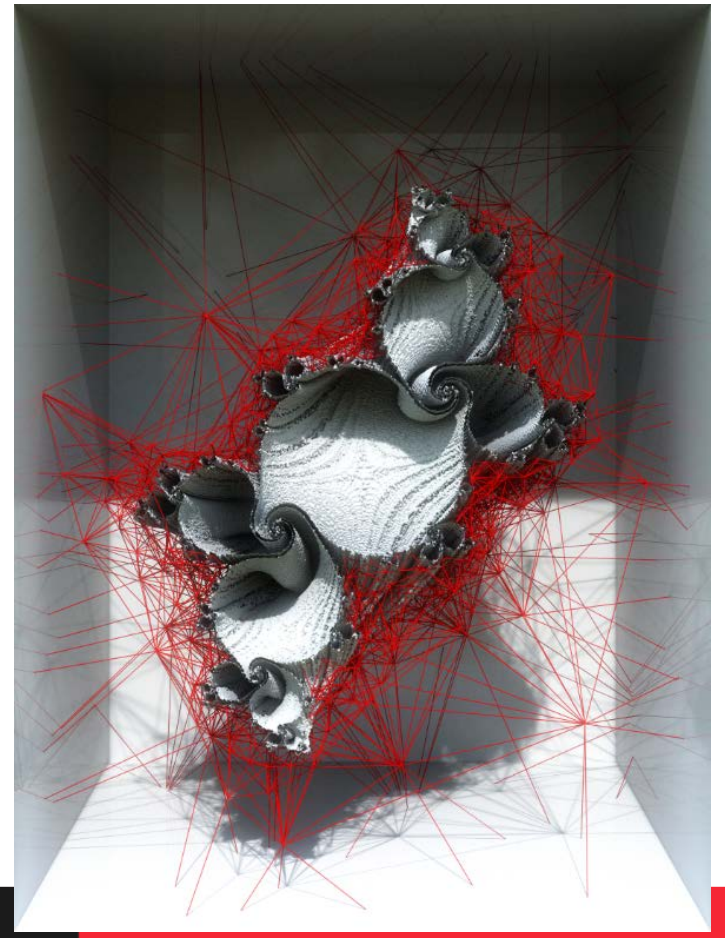
なんちゃって

- ＣＧアート制作
  - フラクタル大好き！
- Delphi 大好き！

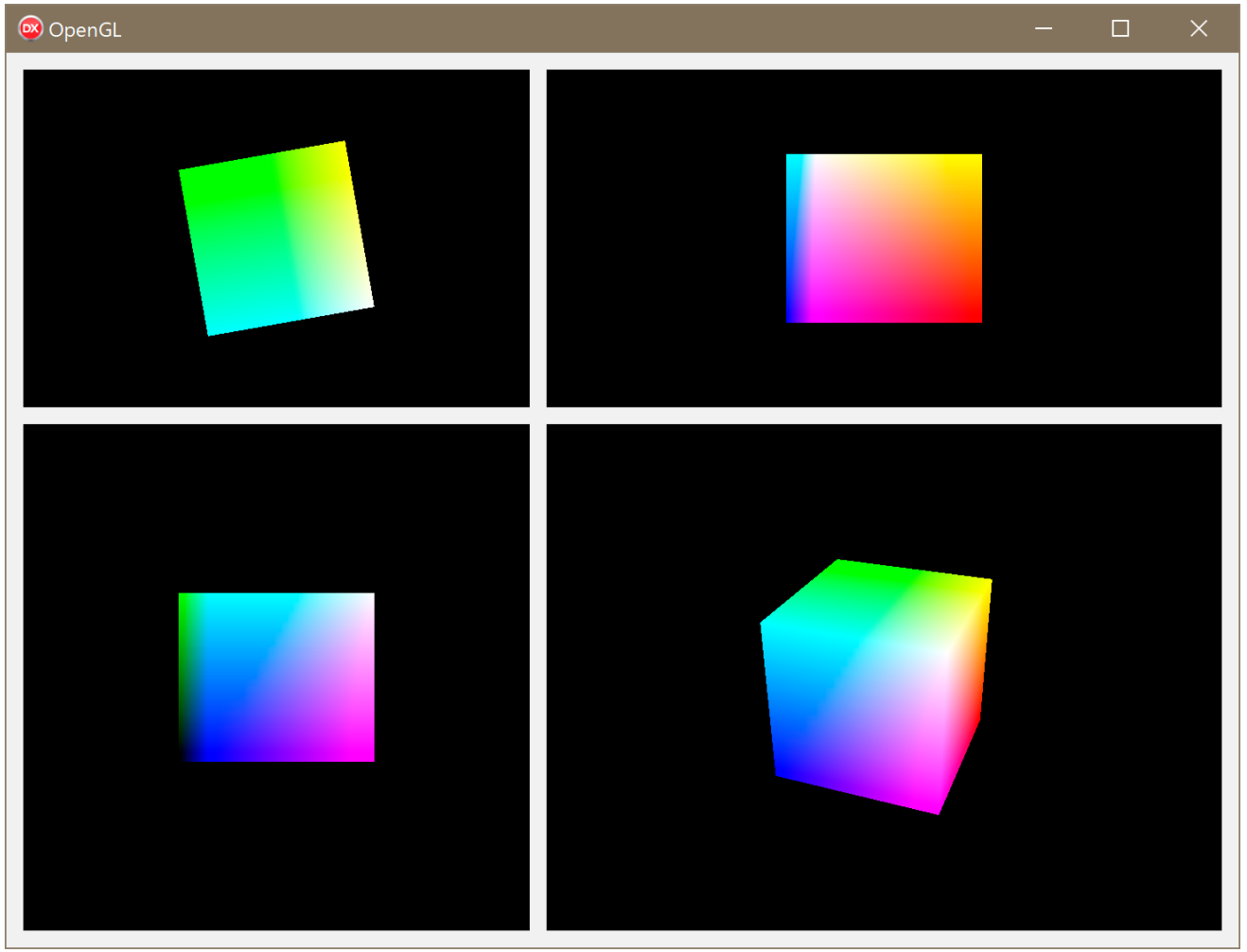






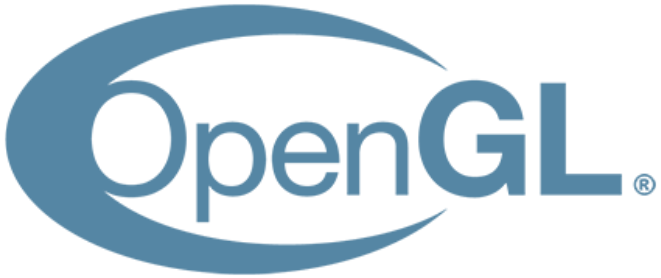


- **OpenGLとは？**

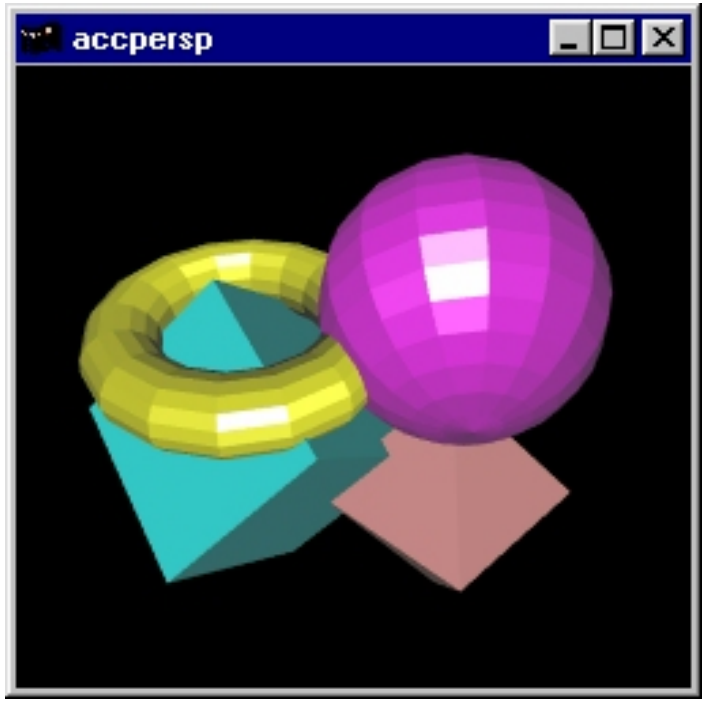


embarcadero DEVELOPER CAMP

OpenGLとは？.規格

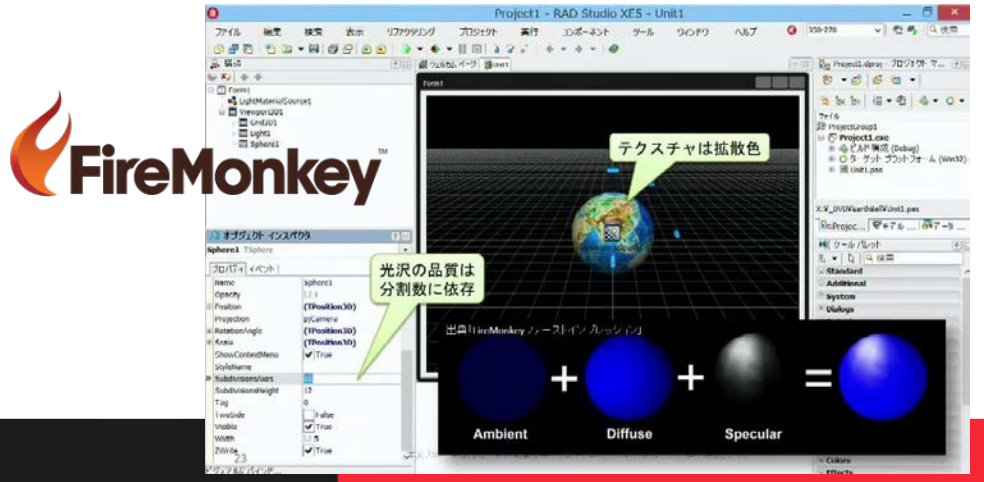


- リアルタイム２Ｄ/３ＤＣＧ用ＡＰＩ
  - GPU：Graphics Processing Unit を活用
    - コンシューマ向けグラフィックボード
    - クリエイター向けグラフィックボード
- Khronos Group が策定
  - `www.khronos.org`
- クロスプラットフォーム
  - Windows, macOS, Android, iOS, Linux…
- C-API ベース
  - ラッパー作りが容易
  - Windows
    - `gl.h` → `Winapi.OpenGL.pas`
    - `glxext.h` → `Winapi.OpenGLext.pas`



OpenGLとは？.ツール

- 高度なゲームを作りたい
  ゲームエンジンを使わざるを得ない
  - Unity
    - C#言語
  - UE：Unreal Engine
    - C++言語 / Blueprint (ノードプログラミング)
- とりあえず勉強したい
  ＧＵＩを想定していない
  - GLUT：OpenGL Utility Toolkit（開発終了）
  - GLFW：OpenGL Framework
- コンポーネントで作りたい
  - FMX：FireMonkey
    - パフォーマンス ＜ 低負荷
  - GLScene
    - Windows, Linux, macOS?



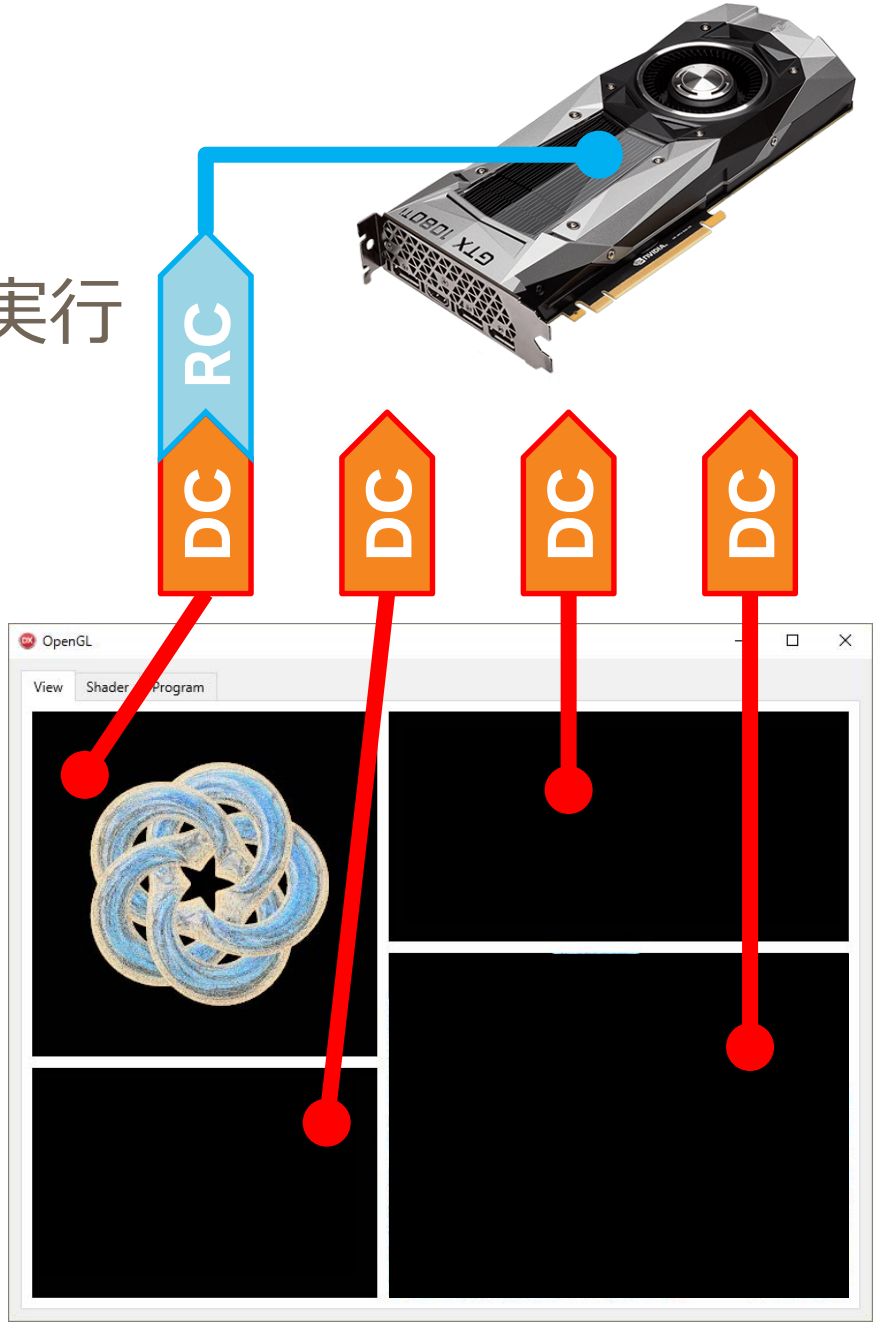
marina.sys.wakayama-u.ac.jp/~tokoi/

OpenGLとは？.資料

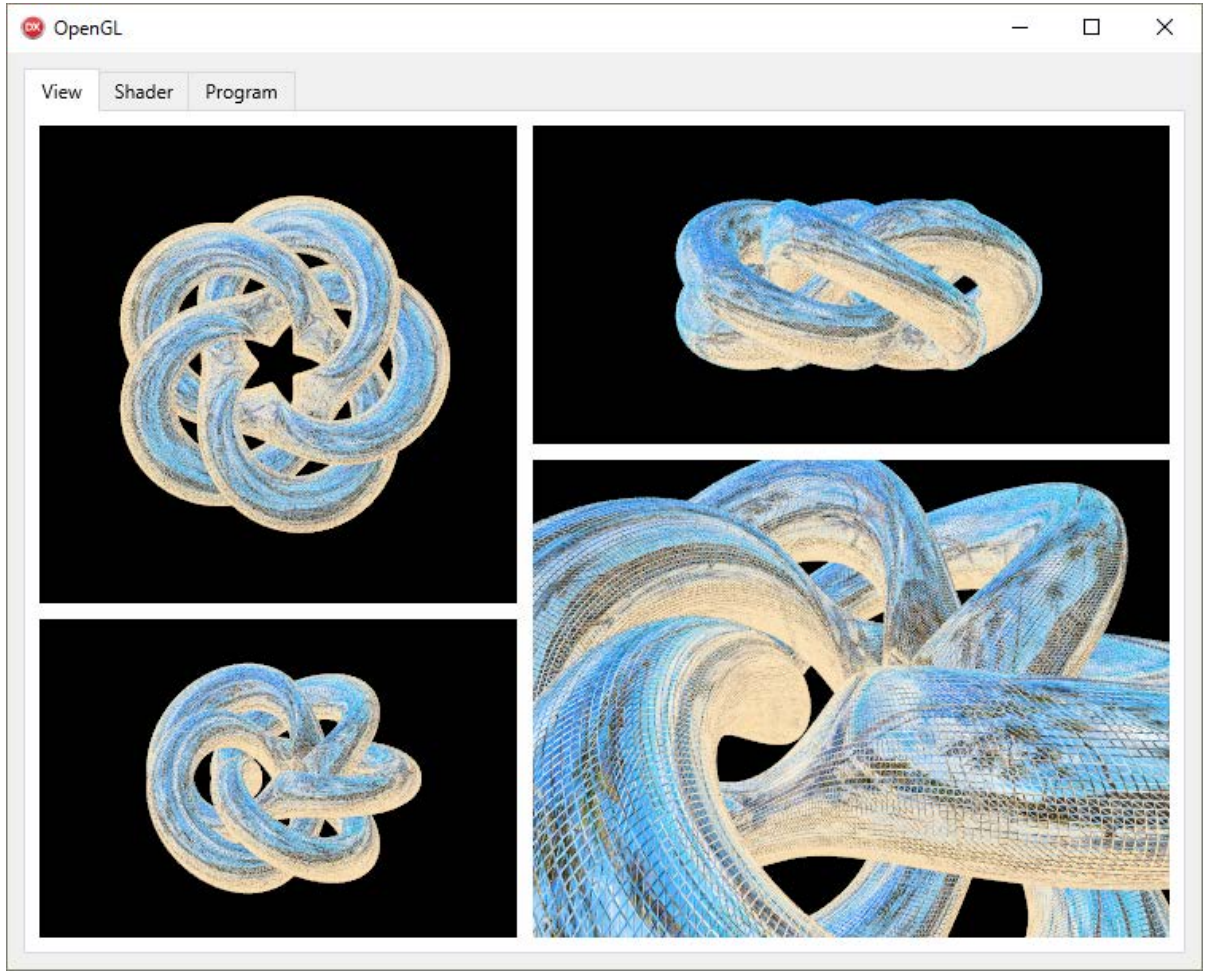
- C/C++ 向けのブログ/書籍が多い
  - ＡＰＩ名は同じなので翻訳は容易
- 昔の資料は役に立たない
  - ＡＰＩ仕様にバージョンが存在
  - レガシーＡＰＩ
    - 性能低下
    - いずれ使えなくなる
- 床井研究室 @ 和歌山大学
  - 床井浩平先生を知らないのはモグリ

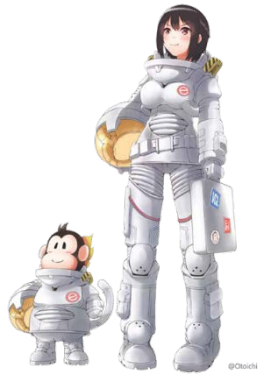
OpenGLとは？.実行

- 描画領域のＤＣにＲＣを結合してＡＰＩを実行
  - ＤＣ：**Device Context**
    - 描画領域のハンドル
    - HWND：ウィンドウハンドルを元に生成
  - ＲＣ：**Rendering Context**
    - OpenGL実行環境のハンドル
    - ＤＣを元に生成
    - 異なるＲＣの間では各種データは共有されない
      wglShareLists 関数によって共有可能
    - 単一のＲＣを複数のＤＣへ使い回すことは可能？
- **wglMakeCurrent** 関数により結合
  - 描画系以外のＡＰＩ実行にも必須
  - プロセス毎に有効化できる結合は１つ
    - マルチスレッド化できない



OpenGLの初期化





embarcadero DEVELOPER CAMP

OpenGLの初期化.手順

① ＤＣの取得
- FMX と VCL で異なる

② 理想のピクセル形式を定義
- TPixelFormatDescriptor 構造体

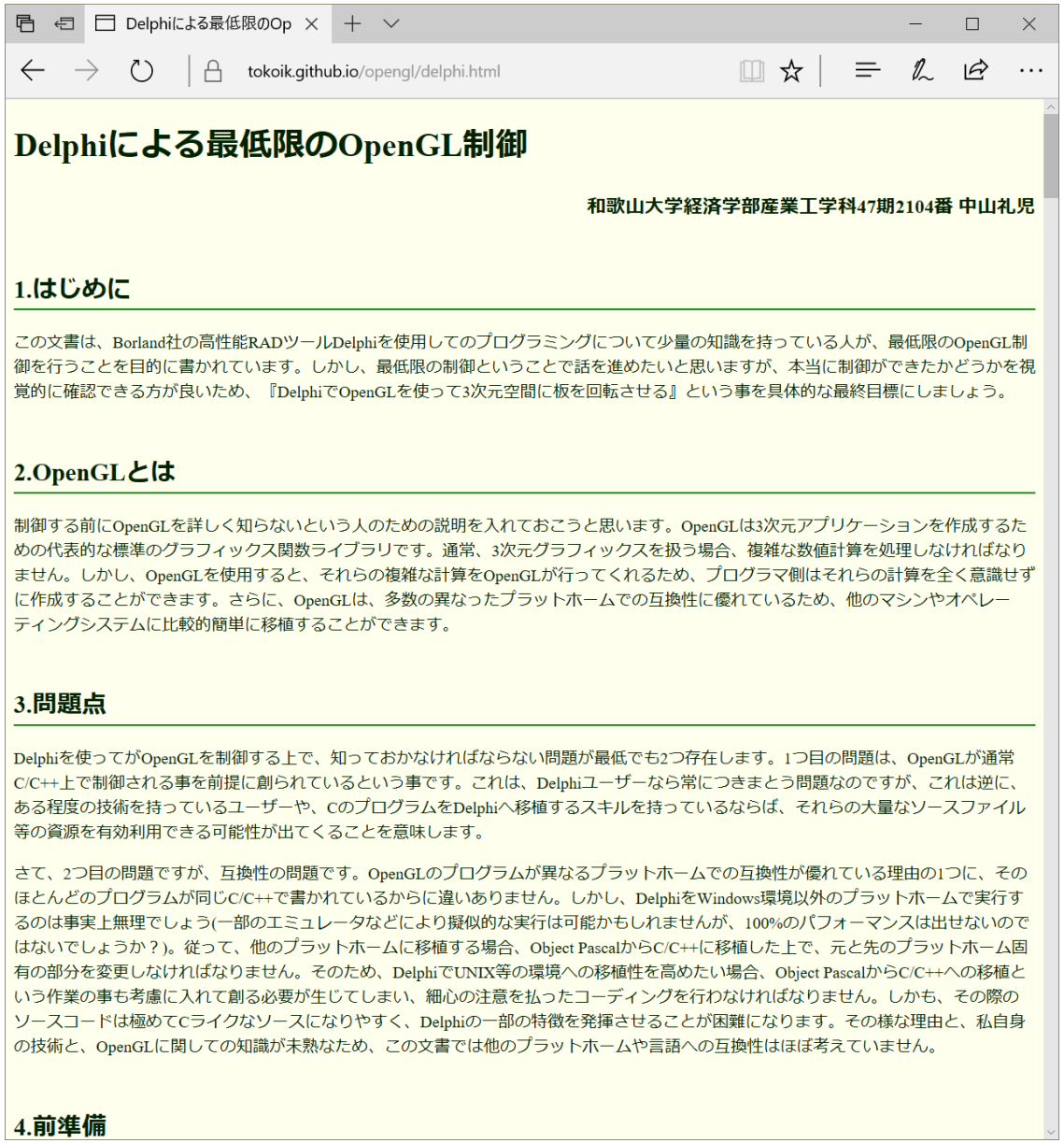
③ 利用可能なピクセル形式を検索
- ChoosePixelFormat 関数

④ ＤＣへ現実ピクセル形式を適用
- SetPixelFormat 関数

⑤ ＲＣの生成
- wglCreateContext 関数

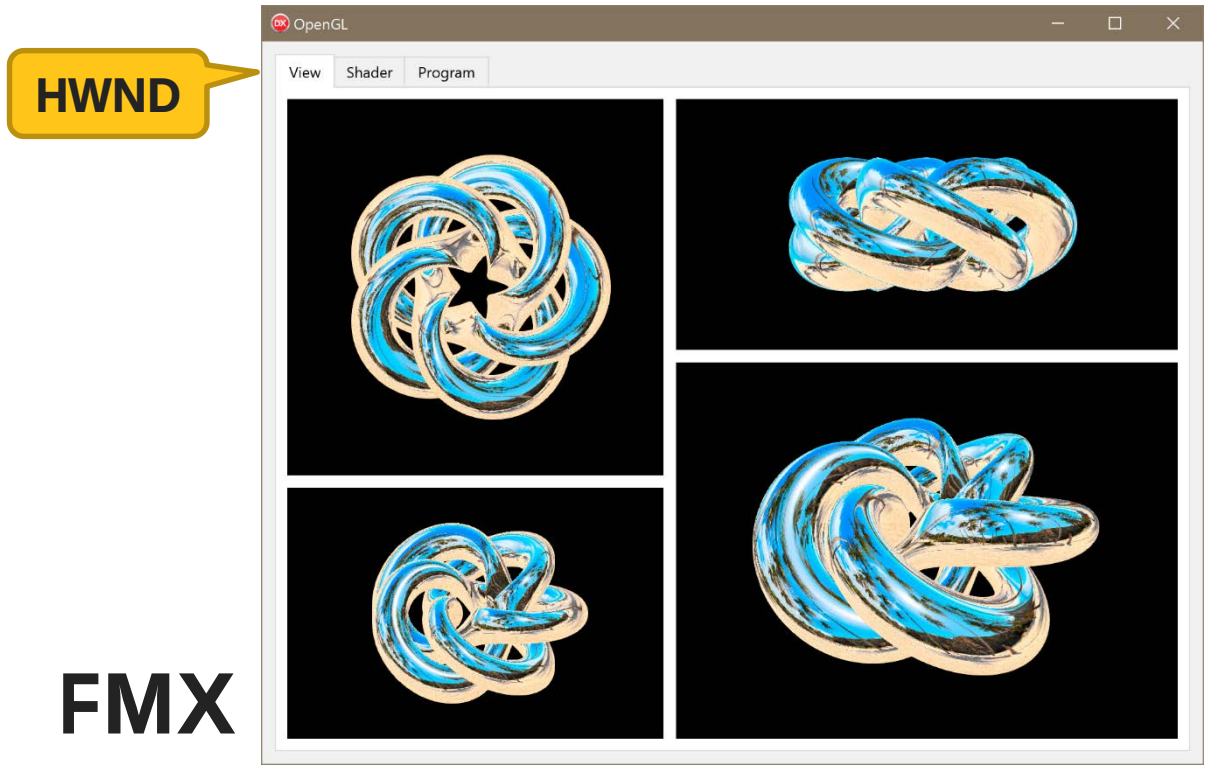
⑥ 拡張ＡＰＩの有効化
- Winapi.OpenGLext.**InitOpenGLext**

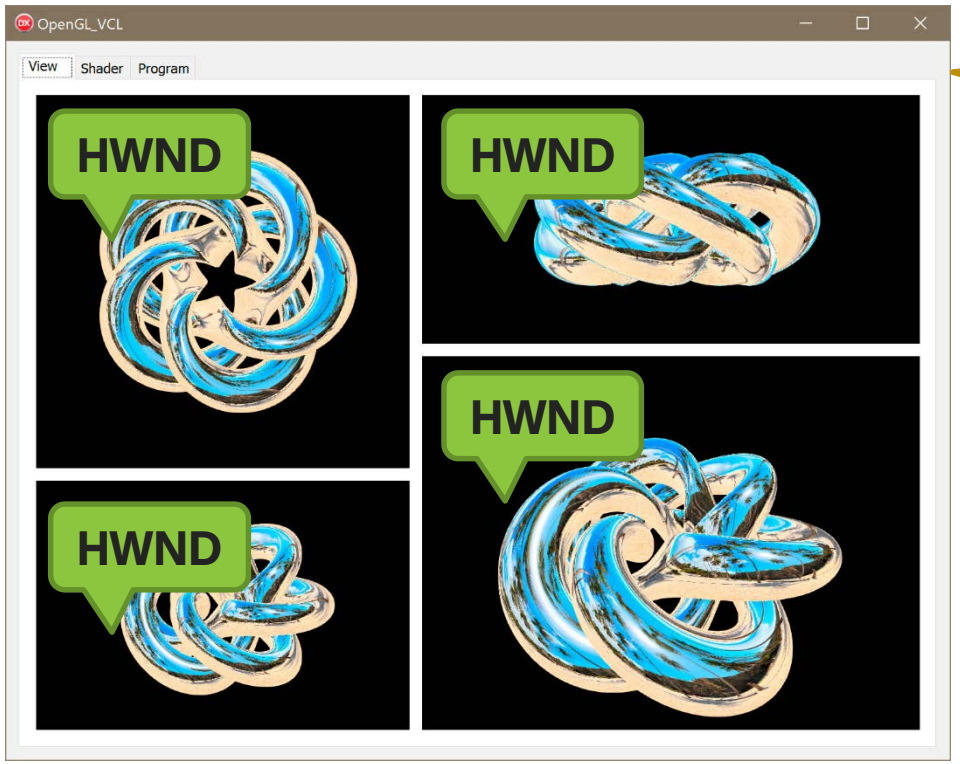
tokoik.github.io/opengl/delphi.html



OpenGLの初期化.①ＤＣの取得

- FMX のコンポーネントはビットマップ
  - ウィンドウ全体のＤＣをＦＭＸが管理
  - コンポーネントの HWND を個別に取得できない





OpenGLの初期化.①ＤＣの取得.ＶＣＬの場合

- TFrame クラスを利用してビューワーコンポーネントを作成
  - **VCL /** LUX.GPU.OpenGL.Viewer.pas
- Vcl.Controls.**TWinControl** を継承するコンポーネント
  - 必ず HWND を持つ
    - **Handle** プロパティにより HWND を取得

```
constructor TGLViewer.Create( AOwner_:TComponent );
begin
      inherited;

      _OnPaint := procedure begin end;

      CreateDC;

      _Viewer := TGLUnifor<TSingleM4>.Create( GL_DYNAMIC_DRAW );
      _Viewer.Count := 1;
end;

destructor TGLViewer.Destroy;
begin
      _Viewer.DisposeOf;

      DestroyDC;

      inherited;
end;
```

```
procedure TGLViewer.CreateDC;
begin
      _DC := GetDC( Handle );

      _OpenGL_.ApplyPixelFormat( _DC );
end;

procedure TGLViewer.DestroyDC;
begin
      ReleaseDC( Handle, _DC );
end;
```

DCを取得

DCを廃棄

OpenGLの初期化.①ＤＣの取得.ＦＭＸの場合

- TFrame クラスを利用してビュワーコンポーネントを作成
  - FMX / LUX.GPU.OpenGL.Viewer.pas
- コンポーネントに縁のないサブウィンドウをはめる
  - TCommonCustomForm クラスから生成
    - FMX.Platform.Win.**WindowHandleToPlatform** 関数で HWND を取得

```
procedure TGLViewer.CreateWindow;
begin
      _Form := TCommonCustomForm.CreateNew( Self );

      with _Form do
      begin
           BorderStyle := TFmxFormBorderStyle.None;

           OnMouseClick := GoMouseClick;
           OnMouseDown  := GoMouseDown ;
           OnMouseMove  := GoMouseMove ;
           OnMouseUp    := GoMouseUp   ;
           OnMouseWheel := GoMouseWheel ;

           HandleNeeded;

           _WND := WindowHandleToPlatform( Handle ).Wnd;
      end;

      SetWindowLong( _WND, GWL_STYLE, WS_CHILD or WS_CLIPSIBLINGS );

      Winapi.Windows.SetParent( _WND, GetRootWND );
end;
```

サブWNDを生成

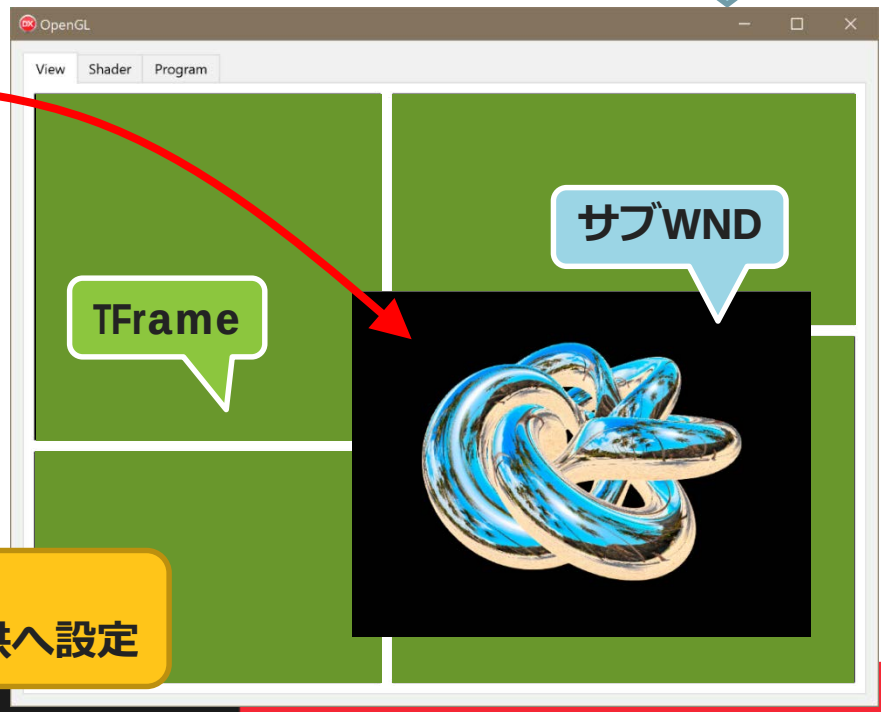
縁を除去

HWNDを強制生成

HWNDを取得

サブWNDのスタイルに
チャイルド属性を付与

サブWNDを
ルートWNDの子供へ設定



OpenGLの初期化.①ＤＣの取得.ＦＭＸの場合.変更検知

- サブWND は TFrame コンポーネントの配下ではない
  - 単に TFrame と同じ位置に**重ねている**だけ
    - TFrame が移動/変形しても サブWND は**追随しない**
  - サイズ変化を検知するには以下のメソッドの override が必須
    - TControl.**DoAbsoluteChanged**
    - TControl.**Resize**

高解像度モニタのスケールを乗算
Scene.GetSceneScale

TFrame の絶対位置と絶対サイズを取得

サブWNDのサイズを変更

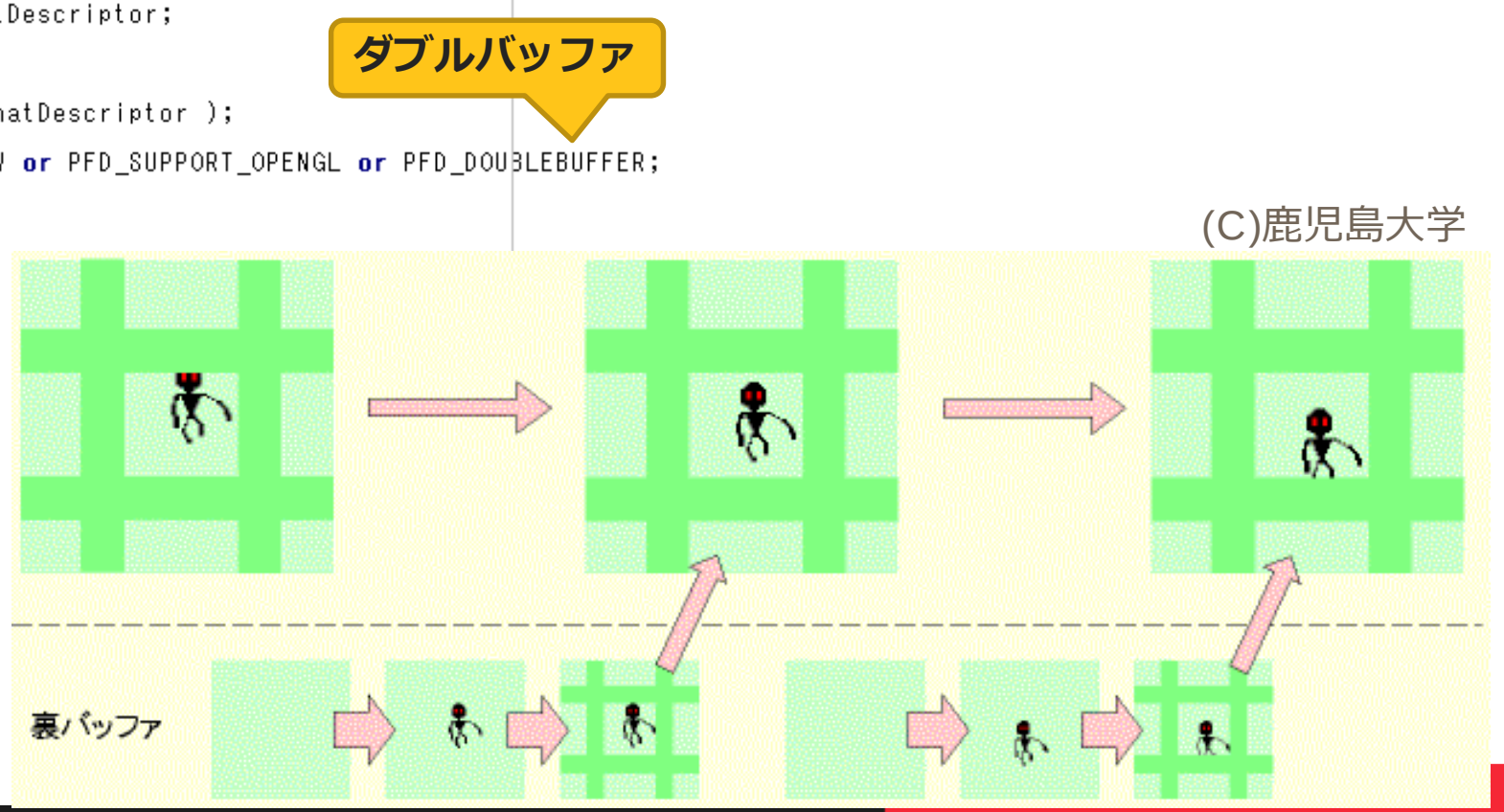
```
procedure TGLViewer.FitWindow;
var
   R :TRectF;
begin
     R := TRectF.Create( LocalToAbsolute( TPointF.Zero ) * Scene.GetSceneScale, Width, Height );

     _Form.Bounds := R.Round;

     if Height < Width then _Viewer[ 0 ] := TSingleM4.Scale( Height / Width, 1, 1 )
                       else
     if Width < Height then _Viewer[ 0 ] := TSingleM4.Scale( 1, Width / Height, 1 )
                       else _Viewer[ 0 ] := TSingleM4.Identify;

end;
```

  - 可視性変化を検知するには以下のメソッドの override が必須
    - TFmxObject.**ParentChanged**
    - TControl.**AncestorVisibleChanged**

OpenGLの初期化.②理想のピクセル形式を定義

- 色のビット数や描画方法を設定
- Winapi.Windows.**TPixelFormatDescriptor** 構造体で定義
  - 実行時のＧＰＵがサポートしている必要はない

```
class function TOpenGL.DefaultPFD :TPixelFormatDescriptor;
begin
     with Result do
     begin
          nSize           := SizeOf( TPixelFormatDescriptor );
          nVersion        := 1;
          dwFlags         := PFD_DRAW_TO_WINDOW or PFD_SUPPORT_OPENGL or PFD_DOUBLEBUFFER;
          iPixelType      := PFD_TYPE_RGBA;
          cColorBits      := 24;
          cRedBits        := 0;
          cRedShift       := 0;
          cGreenBits      := 0;
          cGreenShift     := 0;
          cBlueBits       := 0;
          cBlueShift      := 0;
          cAlphaBits      := 0;
          cAlphaShift     := 0;
          cAccumBits      := 0;
          cAccumRedBits   := 0;
          cAccumGreenBits := 0;
          cAccumBlueBits  := 0;
          cAccumAlphaBits := 0;
          cDepthBits      := 32;
          cStencilBits    := 0;
          cAuxBuffers     := 0;
          iLayerType      := PFD_MAIN_PLANE;
          bReserved       := 0;
          dwLayerMask     := 0;
          dwVisibleMask   := 0;
          dwDamageMask    := 0;
     end;
end;
```



(C)鹿児島大学

OpenGLの初期化.③利用可能なピクセル形式を検索

- Winapi.Windows.**ChoosePixelFormat** 関数で検索
  - ＤＣが扱える範囲で 理想ＰＦＤ に最も近い現実ＰＦＤ を検索
  - 実行デバイスで有効なＰＦＤはＩＤで管理されている

```
procedure TOpenGL.ValidatePFD( const PFD_:TPixelFormatDescriptor );
var
    I :Integer;
begin
    _PFD := PFD_;
    I := ChoosePixelFormat( _DC, @_PFD );
    Assert( I > 0, 'Not found the PixelFormat with a close setting!' );
    ValidatePFI( I );
end;
```

ＤＣ

理想ＰＦＤ

現実ＰＦＤのＩＤ

ＩＤから現実ＰＦＤを取得する

OpenGLの初期化.④ＤＣへピクセル形式を適用

- Winapi.Windows.**SetPixelFormat** 関数で設定
  - 一度設定すると変更できない
    - 一旦ＤＣを廃棄し新規に作成する必要

```
procedure TOpenGL.ApplyPixelFormat( const DC_:HDC );
begin
      Assert( SetPixelFormat( DC_, _PFI, @_PFD ), 'SetPixelFormat() is failed!' );
end;
```

ＤＣ　ＩＤ　ＰＦＤの実データが返る

- Winapi.Windows.**DescribePixelFormat** 関数で取得
  - 設定せずにＰＦＤのＩＤから実データのみを取得

```
procedure TOpenGL.ValidatePFI( const PFI_:Integer );
begin
      _PFI := PFI_;

      Assert( DescribePixelFormat( _DC, _PFI, SizeOf( TPixelFormatDescriptor ), _PFD ),
              'Not found the PixelFormat of the index!' );
end;
```

ＤＣ　ＩＤ　ＰＦＤの実データが返る

OpenGLの初期化.⑤ＲＣの生成

- Winapi.Windows.**wglCreateContext** 関数で生成
- Winapi.Windows.**wglDeleteContext** 関数で廃棄

```
procedure TOpenGL.CreateRC;
begin
      ApplyPixelFormat( _DC );

      _RC := wglCreateContext( _DC );
end;

procedure TOpenGL.DestroyRC;
begin
      wglDeleteContext( _RC );
end;
```

ＤＣへＰＦＤを設定

ＤＣからＲＣを生成

ＲＣを廃棄

OpenGLの初期化.⑥拡張ＡＰＩの有効化

- Winapi.**OpenGLext** ユニットで定義されている拡張ＡＰＩ
  - デフォルトで有効化されていない
- Winapi.OpenGLext.**InitOpenGLext** 関数で初期化する必要あり
  - wglGetProcAddress 関数で各ＡＰＩの関数ポインタを取得
- **wglGetProcAddress** 関数
  - OpenGLのＡＰＩが実行可能な状態でないと実行できない
    - ＤＣとＲＣを結合させてから実行

```
procedure InitOpenGLext;

implementation

procedure InitOpenGLext;
begin
  glDrawRangeElements := wglGetProcAddress('glDrawRangeElements');
  glTexImage3D := wglGetProcAddress('glTexImage3D');
  glTexSubImage3D := wglGetProcAddress('glTexSubImage3D');
  glCopyTexSubImage3D := wglGetProcAddress('glCopyTexSubImage3D');
  glActiveTexture := wglGetProcAddress('glActiveTexture');
  glSampleCoverage := wglGetProcAddress('glSampleCoverage');
  glCompressedTexImage3D := wglGetProcAddress('glCompressedTexImage3D');
```

OpenGLの初期化.②～⑤

- LUX.GPU.OpenGL.**TOpenGL** クラスで行う
- 一時的にコマンド実行を可能とするためのダミーＤＣを用意する
  - **隠しウィンドウ**を作る

```
constructor TOpenGL.Create;
begin
      inherited;
      CreateWindow;
      CreateDC;
      ValidatePFD( DefaultPFD );
      CreateRC;
      BeginGL;
        InitOpenGL;
      EndGL;
      if wglGetCurrentContext = 0 then BeginGL;
end;
```

①隠しウィンドウ生成

②③

④⑤

```
destructor TOpenGL.Destroy;
begin
      if wglGetCurrentContext = _RC then EndGL;
      DestroyRC;
      DestroyDC;
      DestroyWindow;
      inherited;
end;
```

隠しウィンドウ廃棄

OpenGLの初期化.①～⑥

- メインフォームが表示される前に初期化を済ませる
  - ＦＭＸの場合

```
procedure TOepnGL_FMX.CreateWindow;
begin
     _Form := TCommonCustomForm.Create( nil );

     _WND := WindowHandleToPlatform( _Form.Handle ).Wnd;
end;

procedure TOepnGL_FMX.DestroyWindow;
begin
     _Form.DisposeOf;
end;
```

```
initialization //$$$$$$$$$$$$$$$$$$$$
     _OpenGL_ := TOpenGL_VCL.Create;

     InitOpenGLext;
```

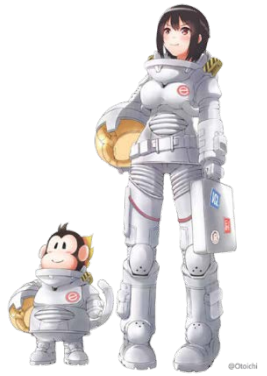
  - ＶＣＬの場合

```
procedure TOpenGL_VCL.CreateWindow;
begin
     _Form := TCustomForm.CreateNew( nil );

     _WND := _Form.Handle;
end;

procedure TOpenGL_VCL.DestroyWindow;
begin
     _Form.Free;
end;
```

```
initialization //$$$$$$$$$$$$$$$$$$$$
     _OpenGL_ := TOepnGL_FMX.Create;

     InitOpenGLext;
```

■コマンドの実行

```
GLViewer1.OnPaint := procedure
begin
      glMatrixMode( GL_PROJECTION );
        glLoadIdentity;
        glOrtho( -2, +2, -2, +2, _N, _F );
      glMatrixMode( GL_MODELVIEW );
        glLoadIdentity;
        glTranslatef( 0, 0, -5 );
        glRotatef( +90, 1, 0, 0 );
        glRotatef( _Angle, 0, 1, 0 );
        DrawShaper;
end;
```



embarcadero DEVELOPER CAMP

コマンドの実行.開始/終了

- 描画したいＤＣとＲＣを結合
  - **wglMakeCurrent** 関数

```
procedure TGLViewer.BeginRender;
begin
      BeginGL;

        glClearColor( 0, 0, 0, 0 );

        glClear( GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT );

        _Viewer.Use( 0{BinP} );
end;

procedure TGLViewer.EndRender;
begin
        _Viewer.Unuse( 0{BinP} );

        glFlush;

      EndGL;

      SwapBuffers( _DC );
end;
```

描画領域のクリア

コマンドキュー強制実行

ダブルバッファ交換

```
procedure TGLViewer.BeginGL;
begin
      _OpenGL_.EndGL;

        wglMakeCurrent( _DC, _OpenGL_.RC );
end;

procedure TGLViewer.EndGL;
begin
        wglMakeCurrent( _DC, 0 );

      _OpenGL_.BeginGL;
end;
```

TGLViewer のＤＣと
隠しＷＮＤのＲＣを結合

TGLViewer のＤＣを外す

```
procedure TOpenGL.BeginGL;
begin
      wglMakeCurrent( _DC, _RC );
end;

procedure TOpenGL.EndGL;
begin
      wglMakeCurrent( _DC, 0 );
end;
```

隠しＷＮＤのＤＣを外す

コマンドの実行.OnPaint イベント

- ＦＭＸの場合
  - Paint メソッドで描画
- ＶＣＬの場合
  - WMPaint メソッドで描画
  - WMEraseBkgnd メソッドを無効化

```
GLViewer1.OnPaint := procedure
begin
      glMatrixMode( GL_PROJECTION );
        glLoadIdentity;
        glOrtho( -2, +2, -2, +2, _N, _F );
      glMatrixMode( GL_MODELVIEW );
        glLoadIdentity;
        glTranslatef( 0, 0, -5 );
        glRotatef( +90, 1, 0, 0 );
        glRotatef( _Angle, 0, 1, 0 );
        DrawShaper;
end;
```

OpenGLコマンド

高画質モニタの
スケールを考慮した実ピクセル数

```
procedure TGLViewer.Paint;
begin
      BeginRender;

        with GetPixSiz do glViewport( 0, 0, Width, Height );

        if Assigned( _Camera ) then _Camera.Render;

        if Assigned( _OnPaint ) then _OnPaint;

      EndRender;
end;
```

描画範囲の設定

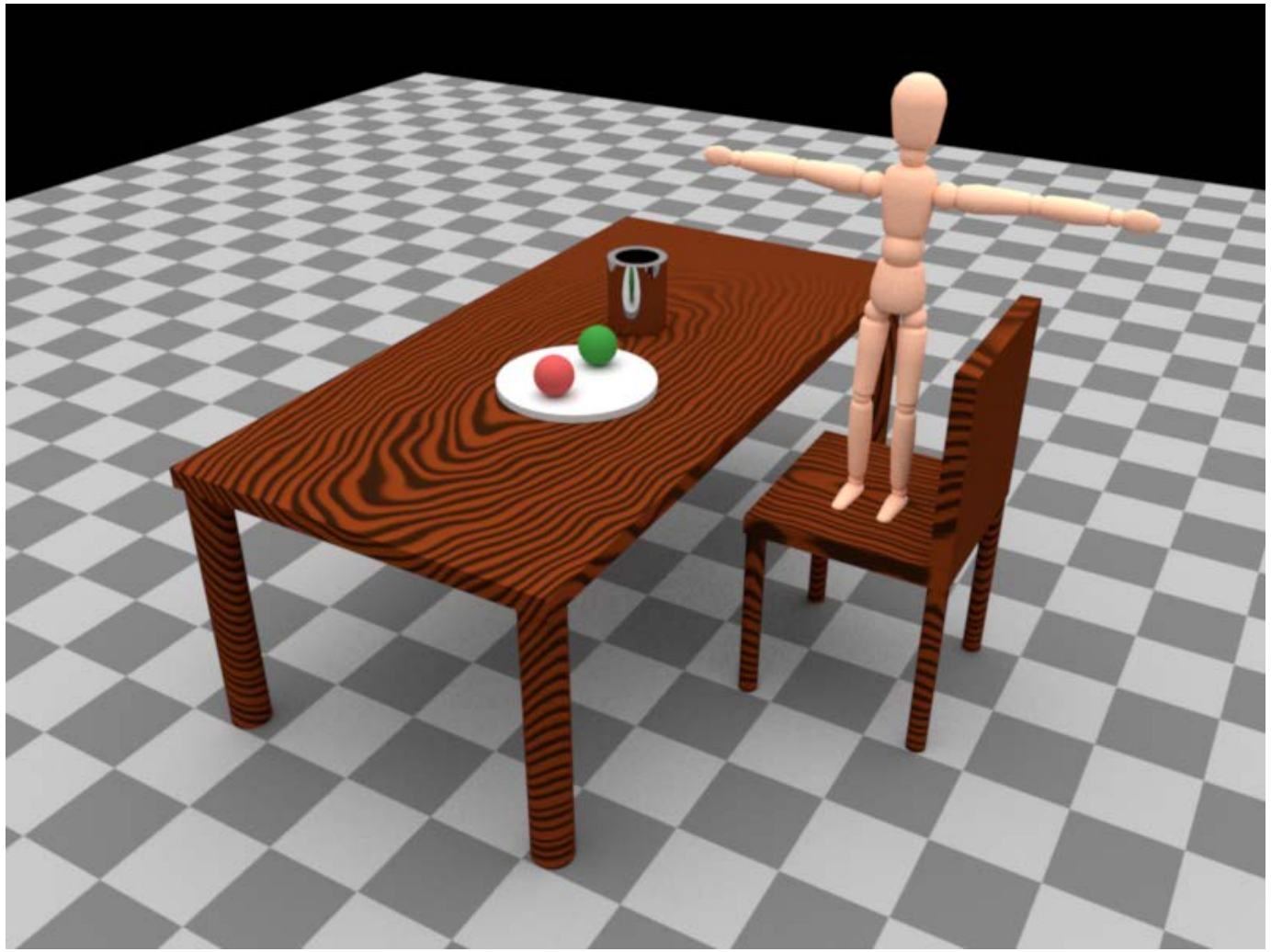
OnPaint 実行

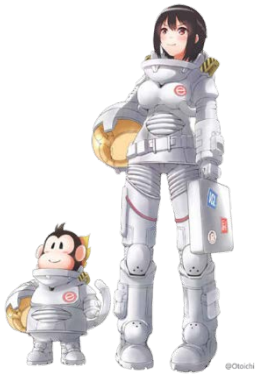
```
procedure TGLViewer.WMEraseBkgnd( var Message_:TWmEraseBkgnd );
begin
      ///// 背景描画を無効化
end;

procedure TGLViewer.WMPaint( var Message_:TWMPaint );
begin
      inherited;

      BeginRender;

        glViewport( 0, 0, ClientWidth, ClientHeight );

        if Assigned( _Camera ) then _Camera.Render;

        if Assigned( _OnPaint ) then _OnPaint;

      EndRender;
end;
```

描画範囲の設定

OnPaint 実行

シーンの構築

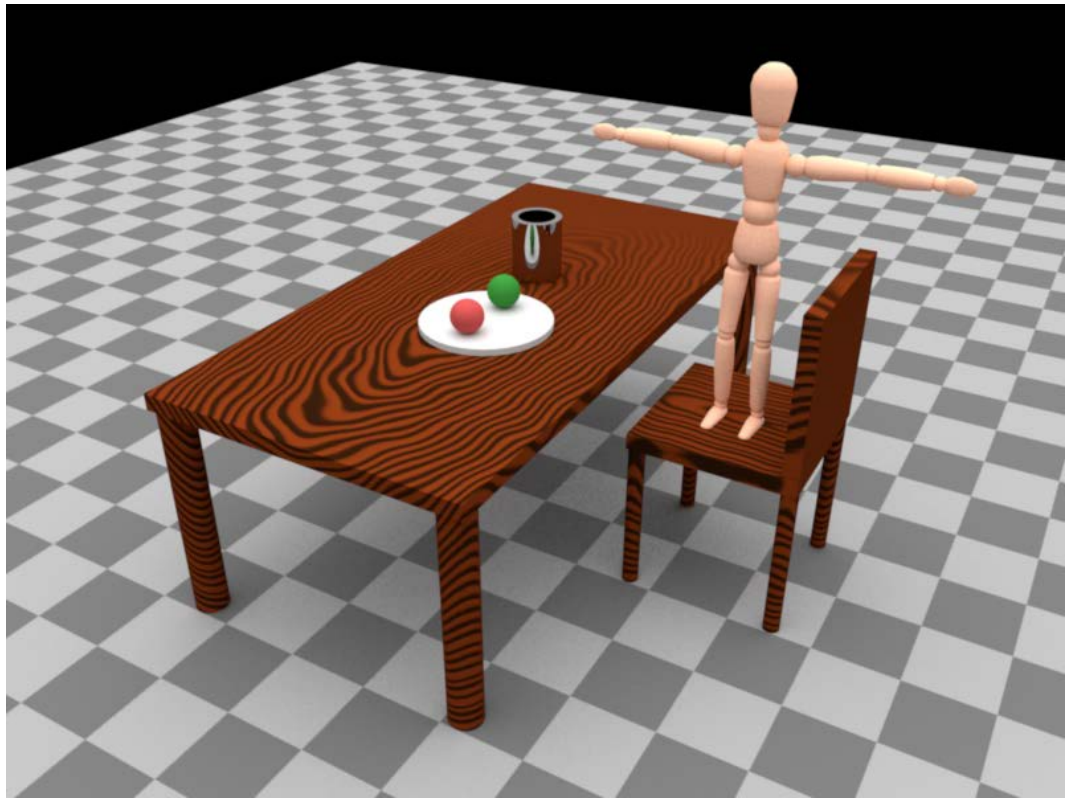


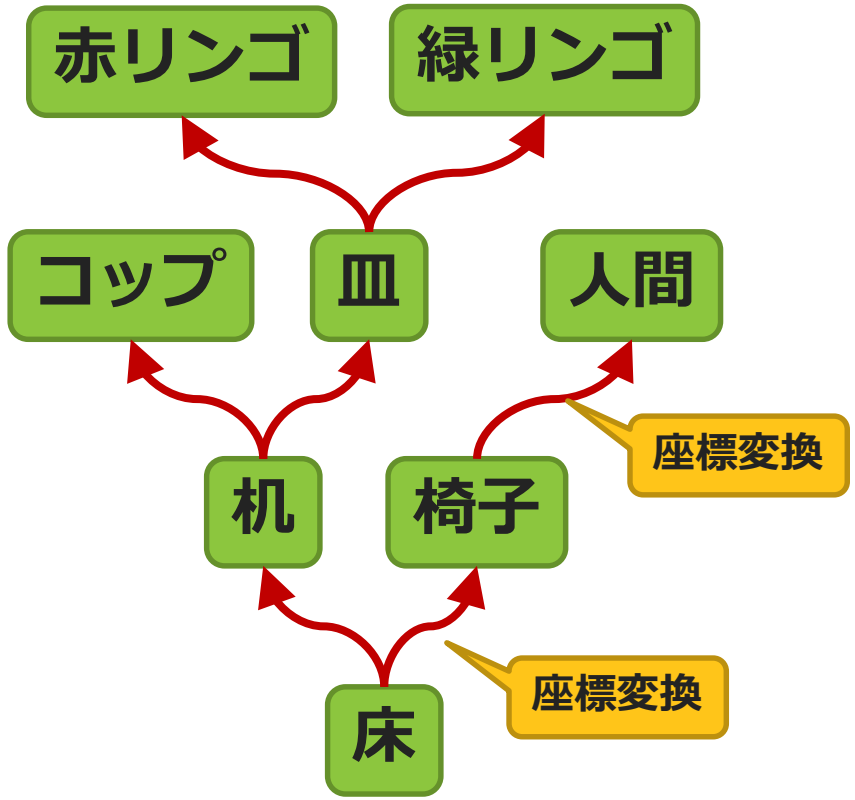


embarcadero DEVELOPER CAMP

シーンの構築.シーングラフ

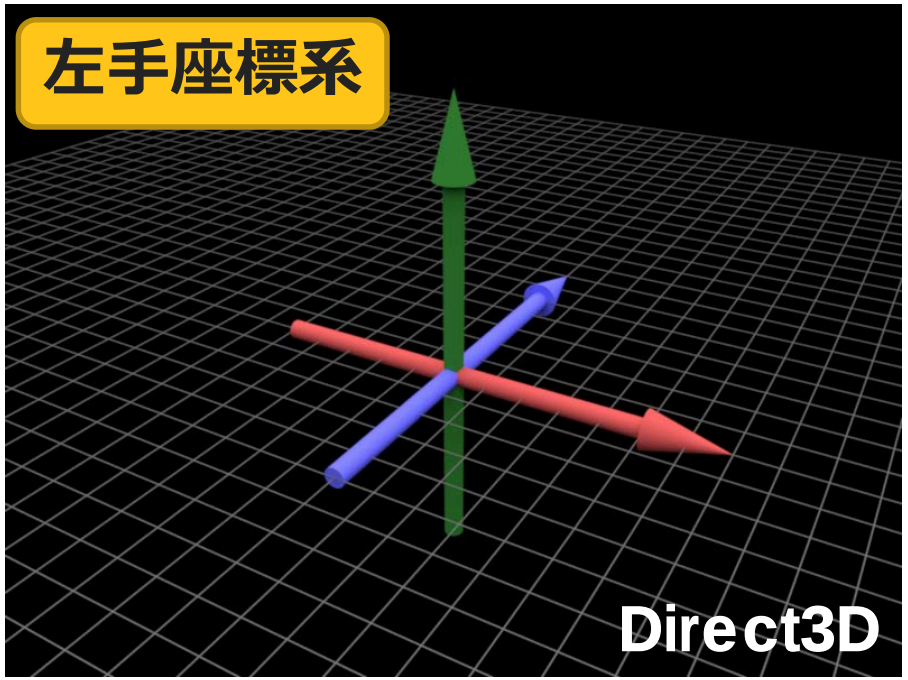
- シーン構造をツリー構造で表現
  - 物体はノード
  - パスを経る旅に位置/姿勢が変更されると考える

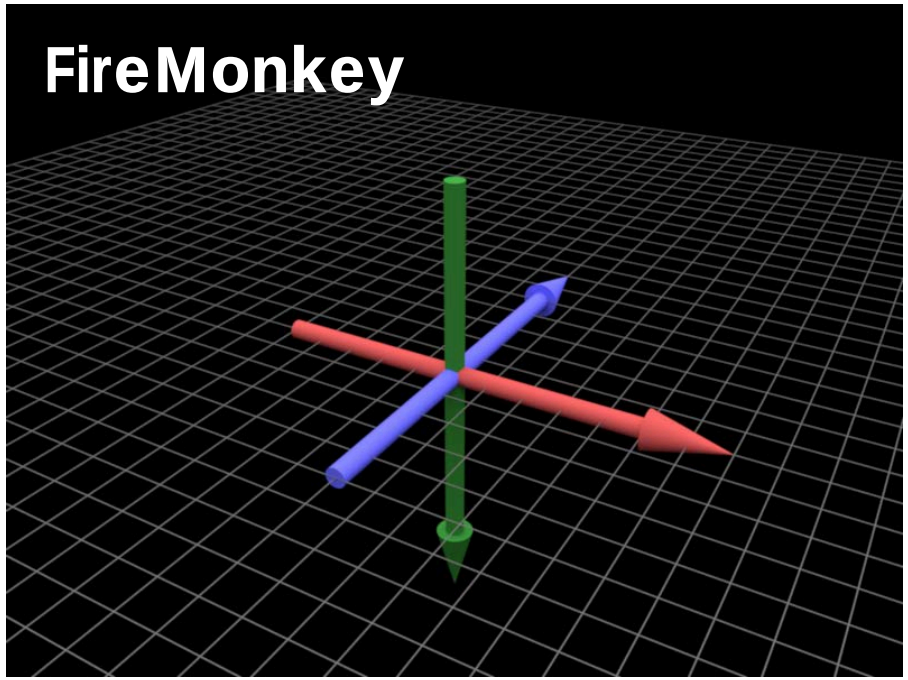


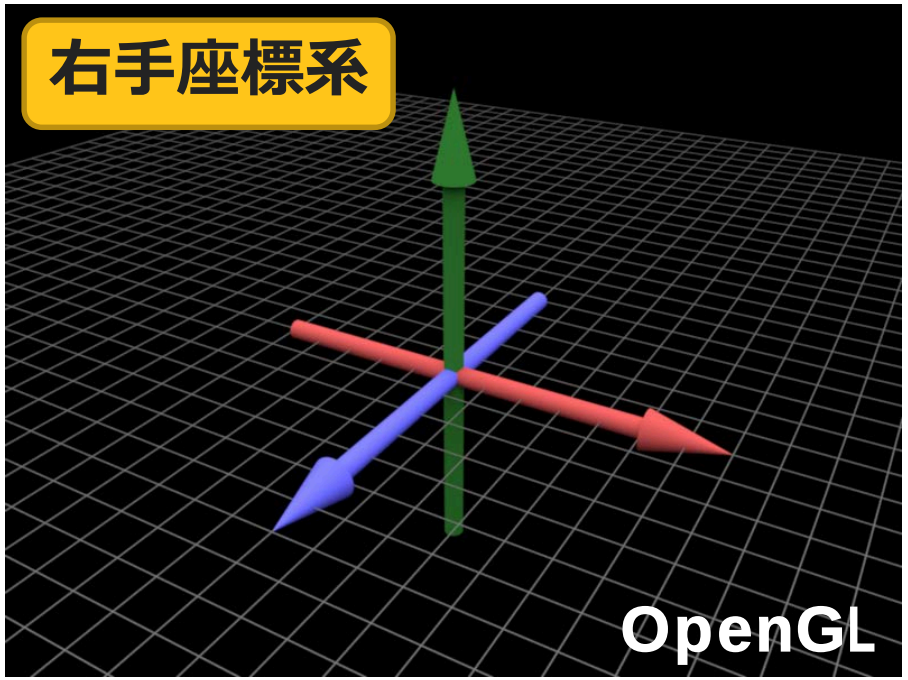


シーンの構築.座標系

- OpenGL は右手座標系
  - X： 親指
  - Y： 人差し指
  - Z： 中指







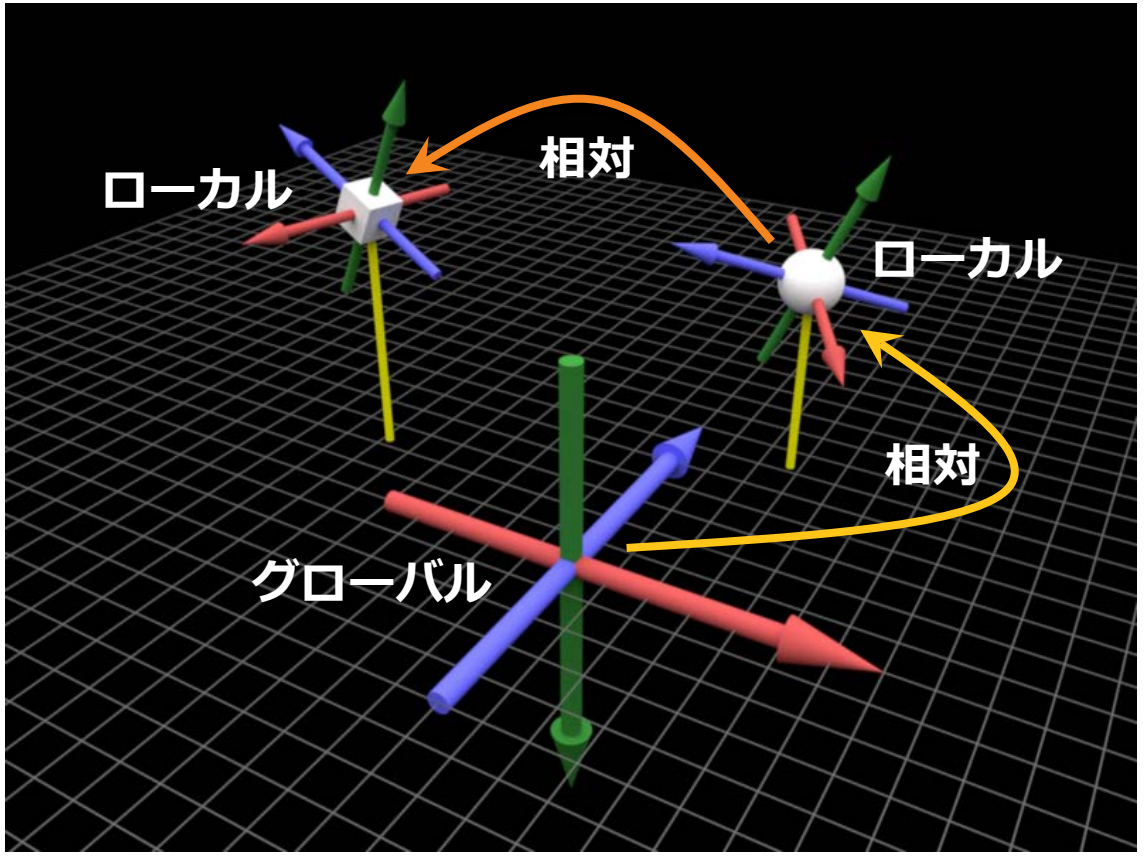
シーンの構築.座標

- グローバル（ワールド）座標系
  - ただひとつの絶対基準

- ローカル（オブジェクト）座標系
  - 物体の姿勢を表す

- 絶対座標
  - グローバル座標系から見た絶対位置

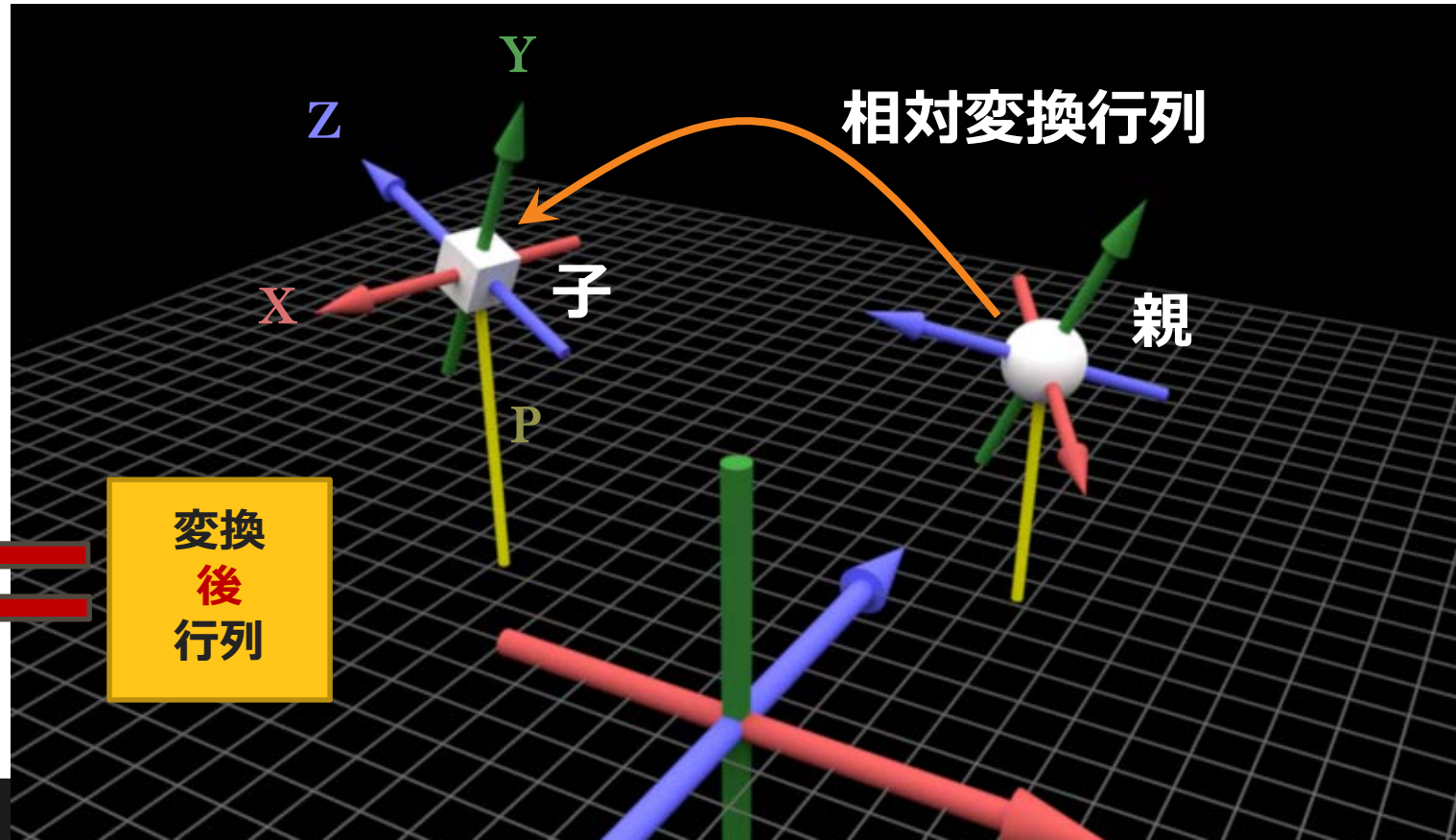
- 相対座標
  - ローカル座標系から見た相対位置



シーンの構築.同時座標系

- 座標変換は行列（４×４）の演算で表現できる
  - ローカル座標系の基底ベクトルを並べたもの
  - 複数の変換を合成して１つの行列で表せる

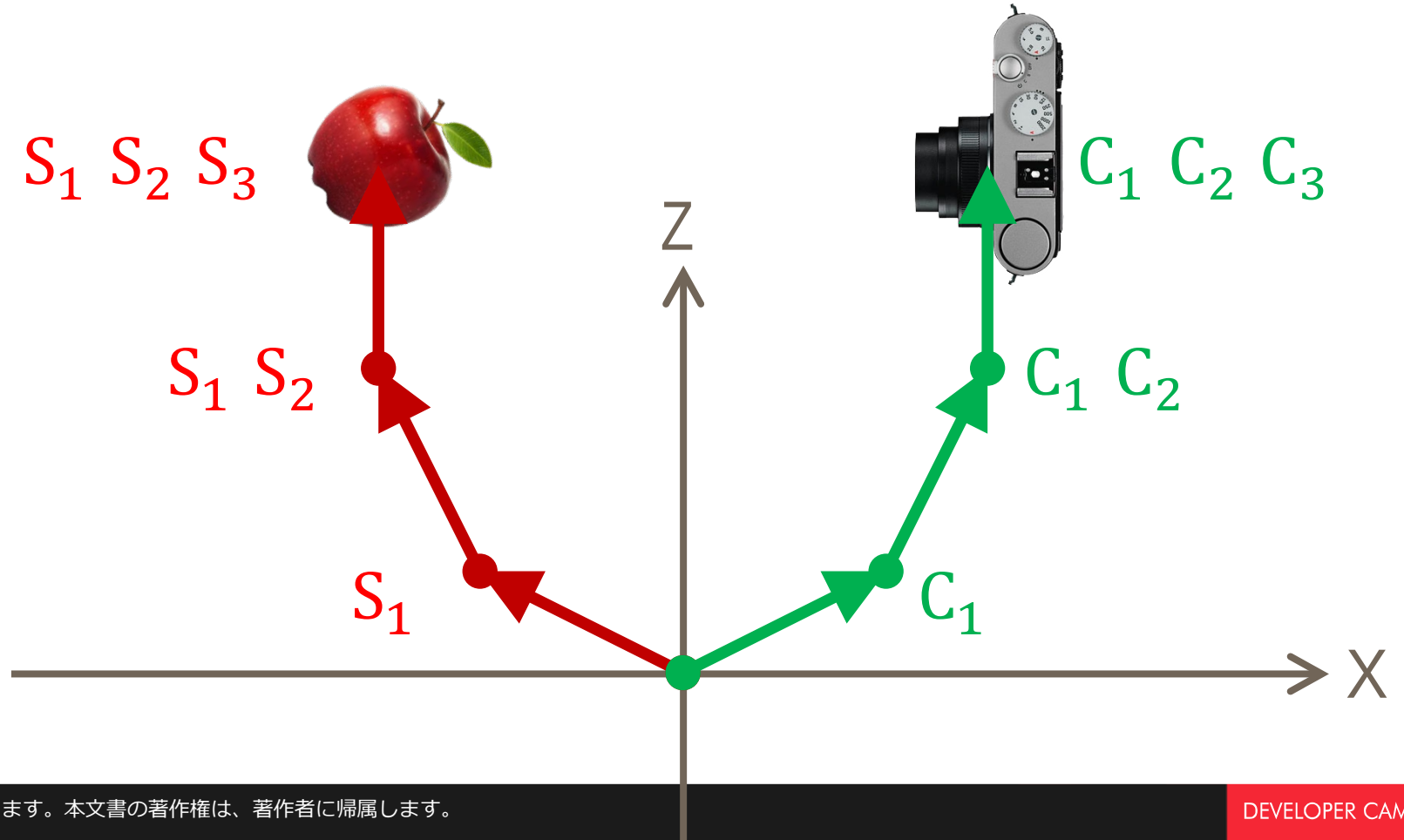
$$\begin{bmatrix} X_X & Y_X & Z_X & P_X \\ X_Y & Y_Y & Z_Y & P_Y \\ X_Z & Y_Z & Z_Z & P_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} A_X \\ A_Y \\ A_Z \\ 1 \end{bmatrix} = \begin{bmatrix} B_X \\ B_Y \\ B_Z \\ 1 \end{bmatrix}$$



子 基準 変換 × 変換 前 行列 × 親 基準 変換 = 変換 後 行列

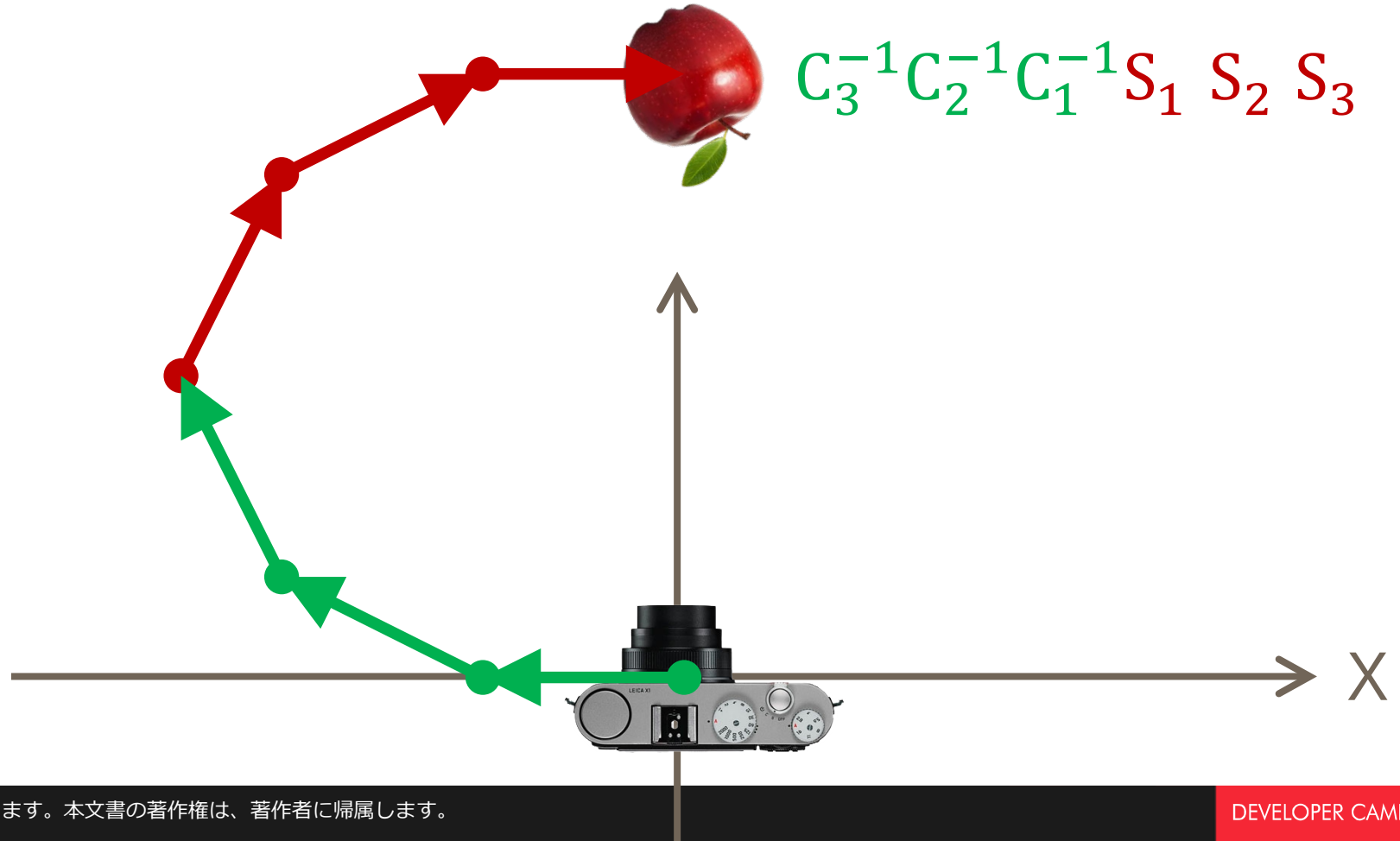
シーンの構築.カメラ

- カメラも位置と姿勢を持つ物体の一種である



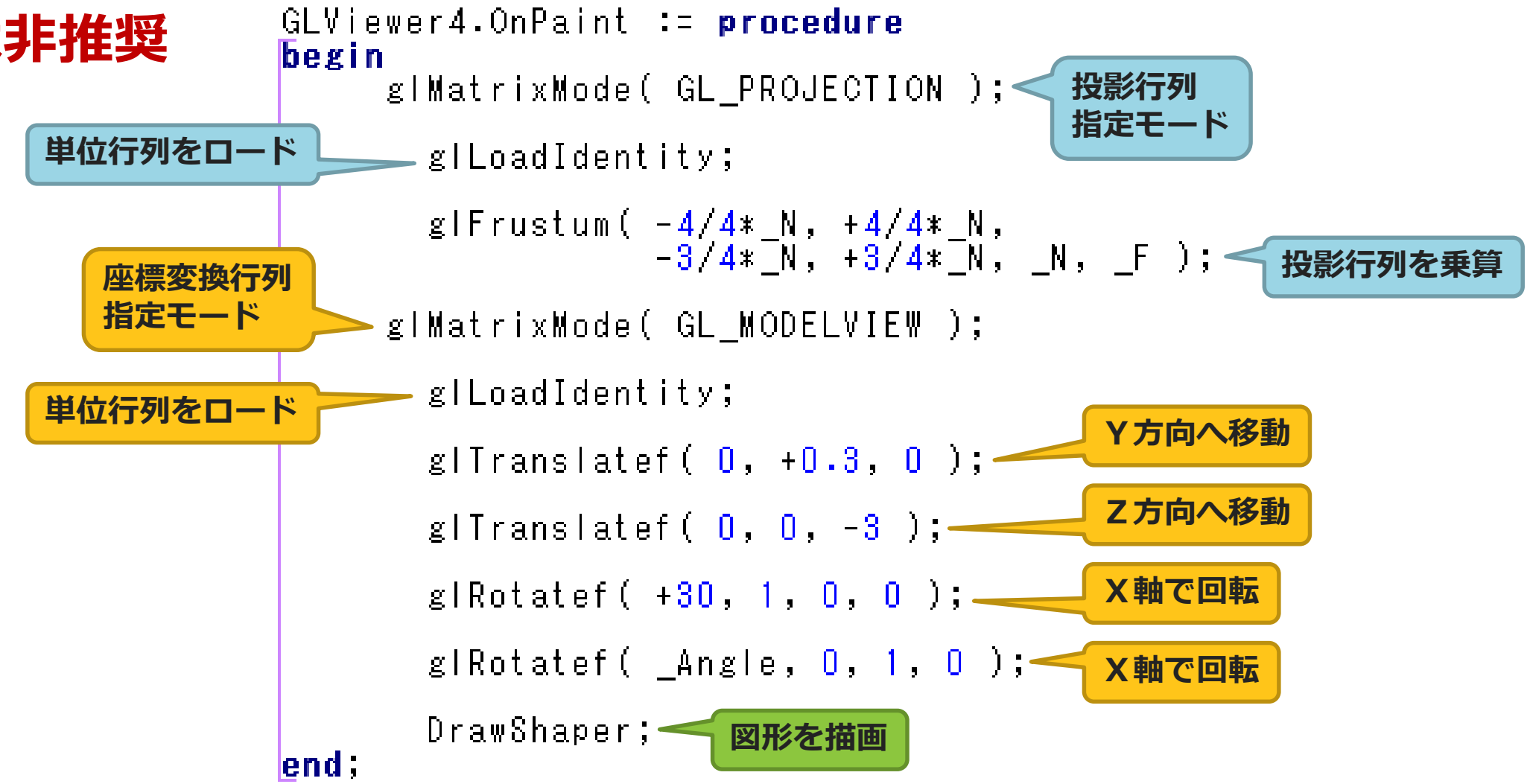
シーンの構築.カメラ.視界

- OpenGL はカメラから見た風景しか描けない
  - 物体にカメラの**逆行列**を左から掛ける



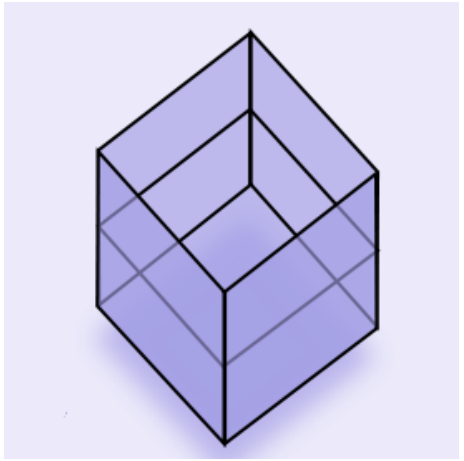
シーンの構築.カメラ.設定

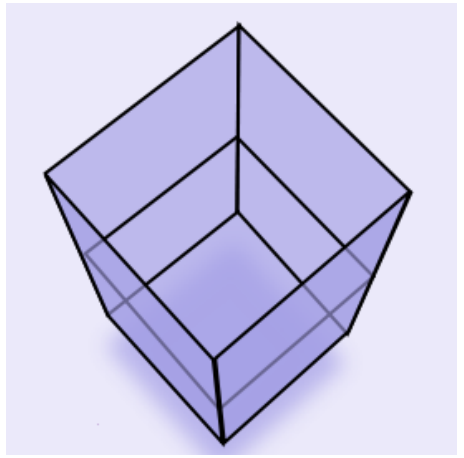
- 現在は非推奨

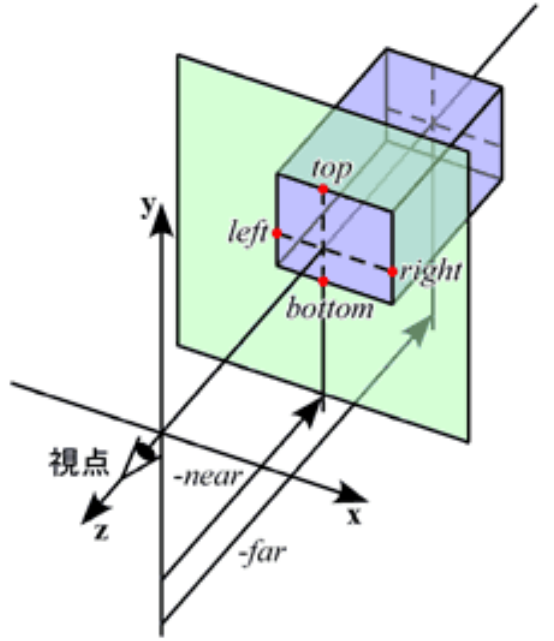


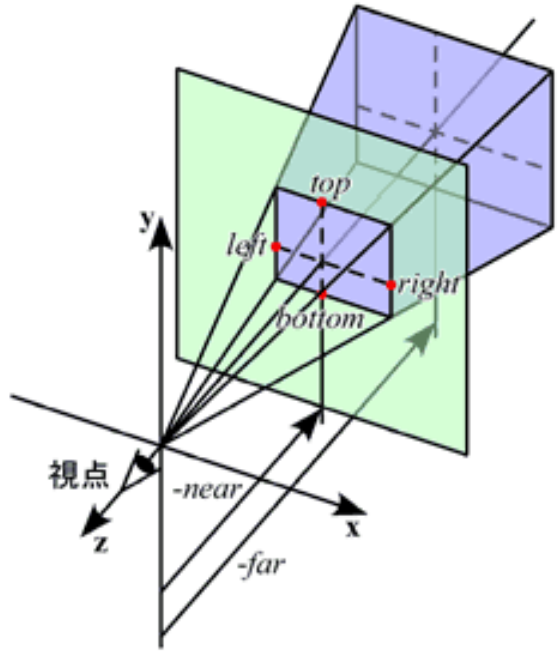
シーンの構築.カメラ.投影方法

- 風景をスクリーンへ投影するのも座標変換
  - **投影行列**によって指定可能
- 平行投影
  - 寸法や平行線が歪まない図面的表示
  - **glOrtho** 関数で行列生成
- 透視投影
  - 遠近法を加味したリアルな表示
  - **glFrustum** 関数で行列生成



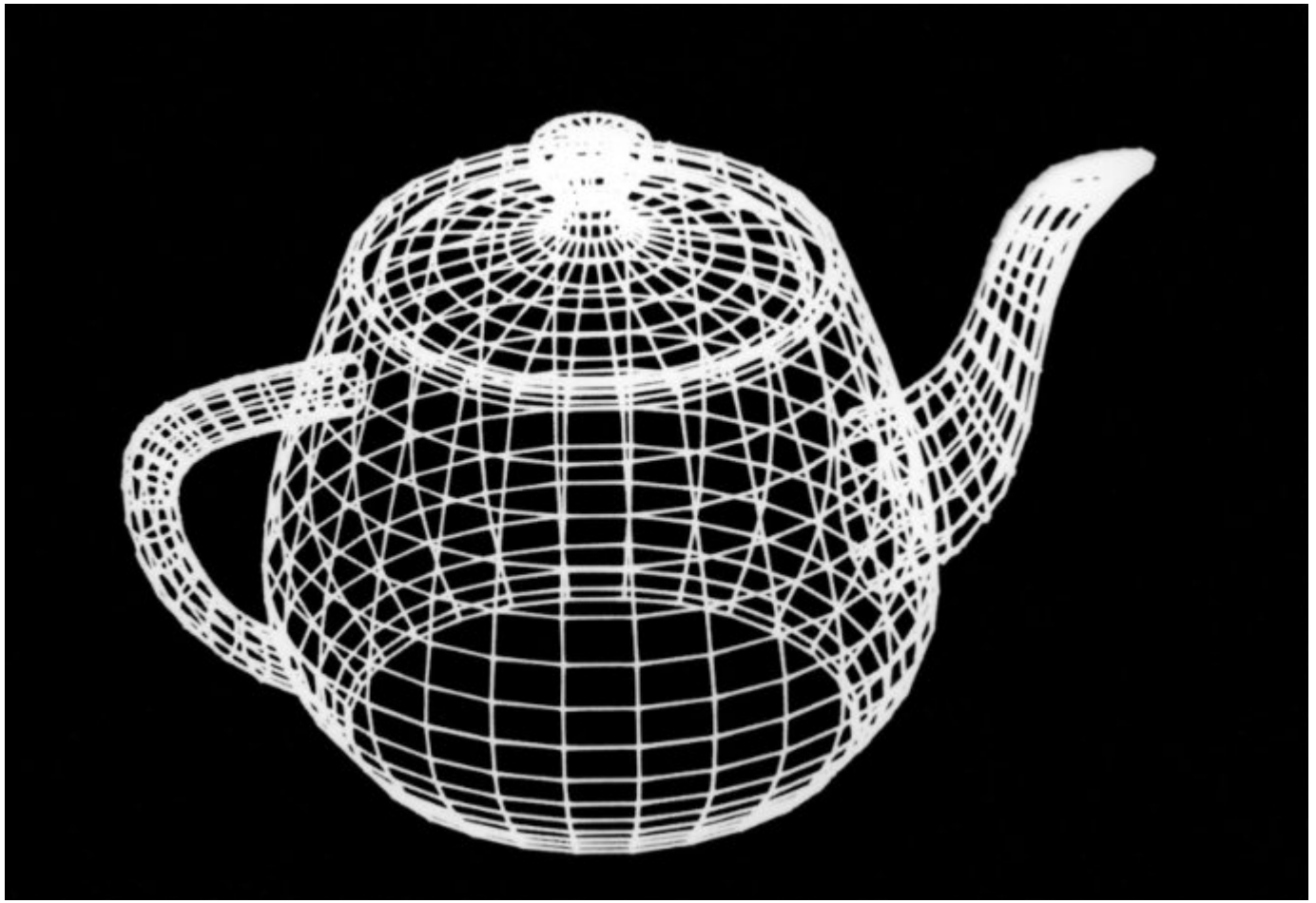


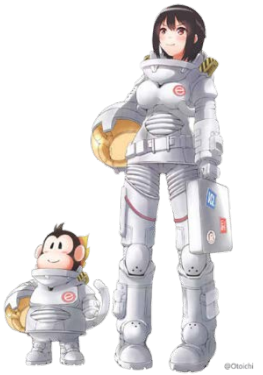




marina.sys.wakayama-u.ac.jp/~tokoi/?date=20090829

▪物体の定義

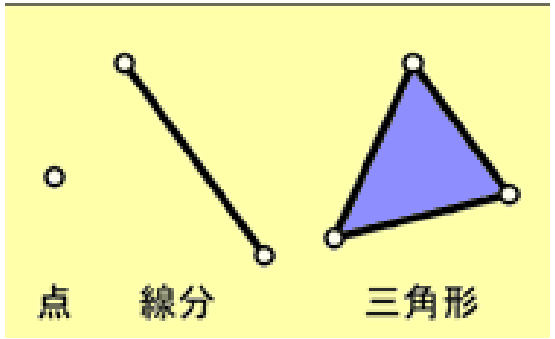


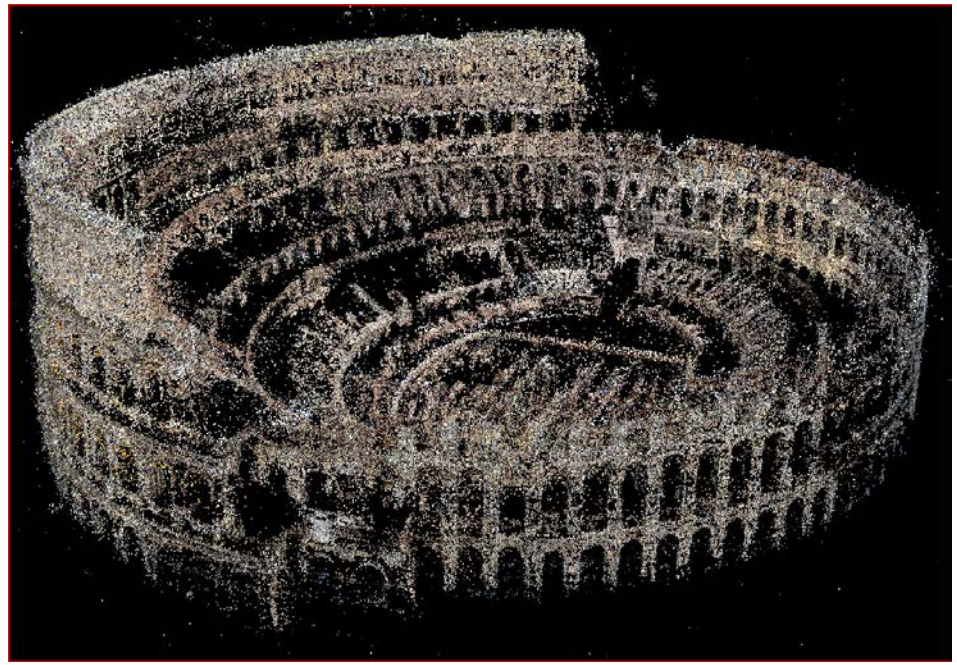


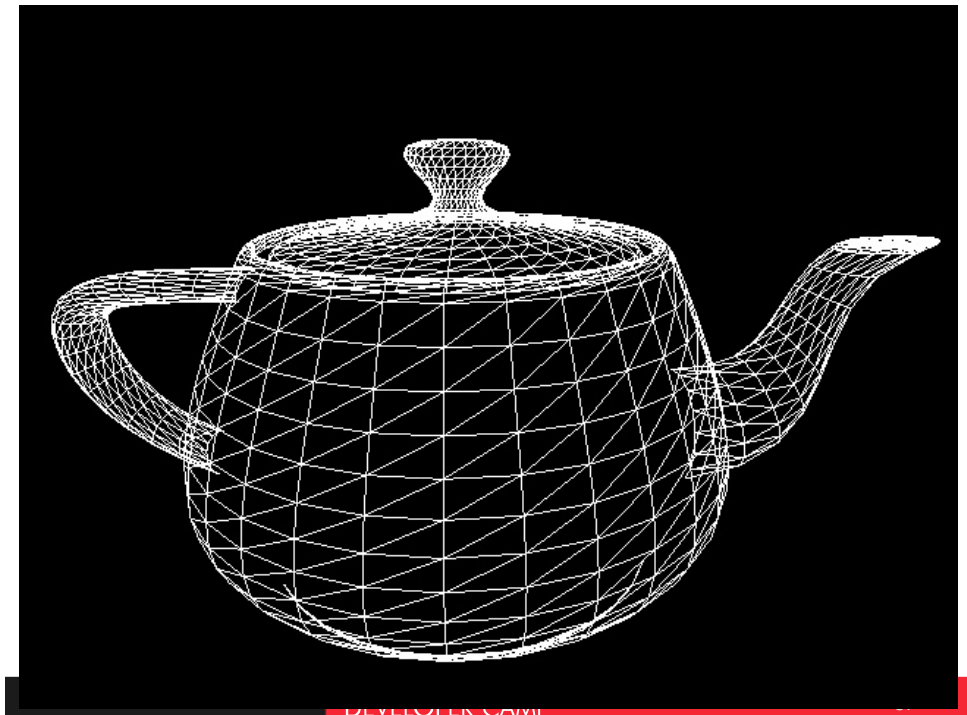
embarcadero®
DEVELOPER CAMP

物体の定義.構成要素

- 大量に描くことで様々な形状を表現
  点 と 線分 と 三角形 しか描けない
  - **Point Cloud** モデル
  - **Wireframe** モデル
  - **Polygon Mesh** モデル

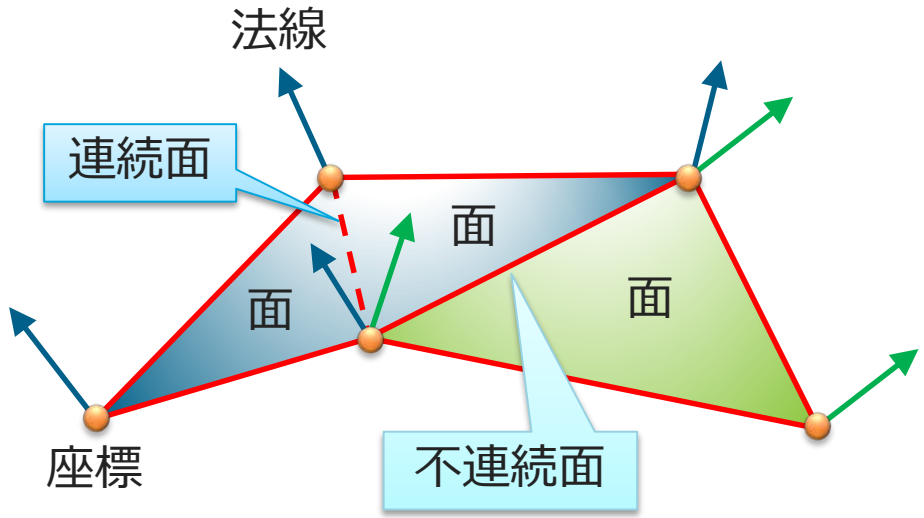


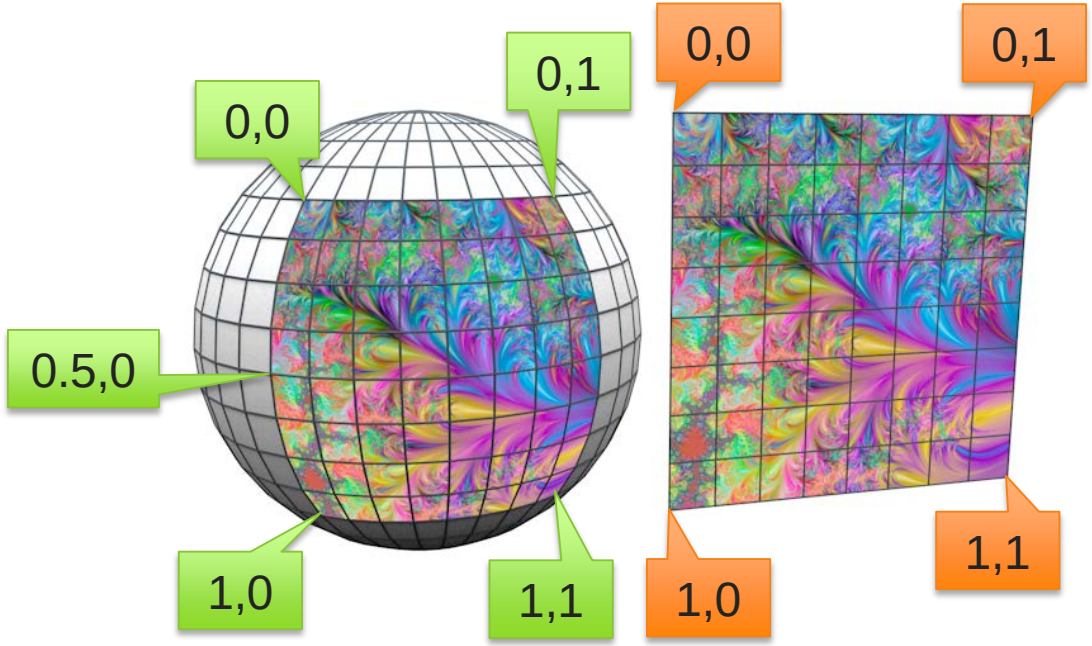




物体の定義.頂点の情報

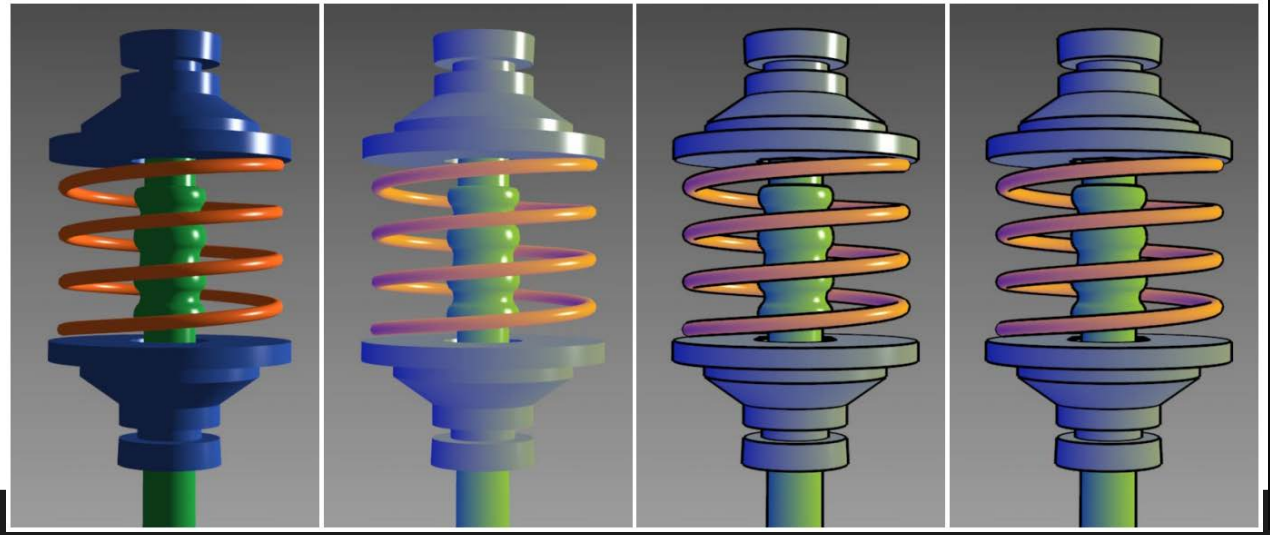
- 頂点の情報で物体は定義される
  - 位置
    - ３次元座標
  - 法線ベクトル
    - 面に垂直な３次元ベクトル
  - 色
    - 基本的にテクスチャで代替される
  - テクスチャ座標
    - 貼り付ける画像上の２次元座標

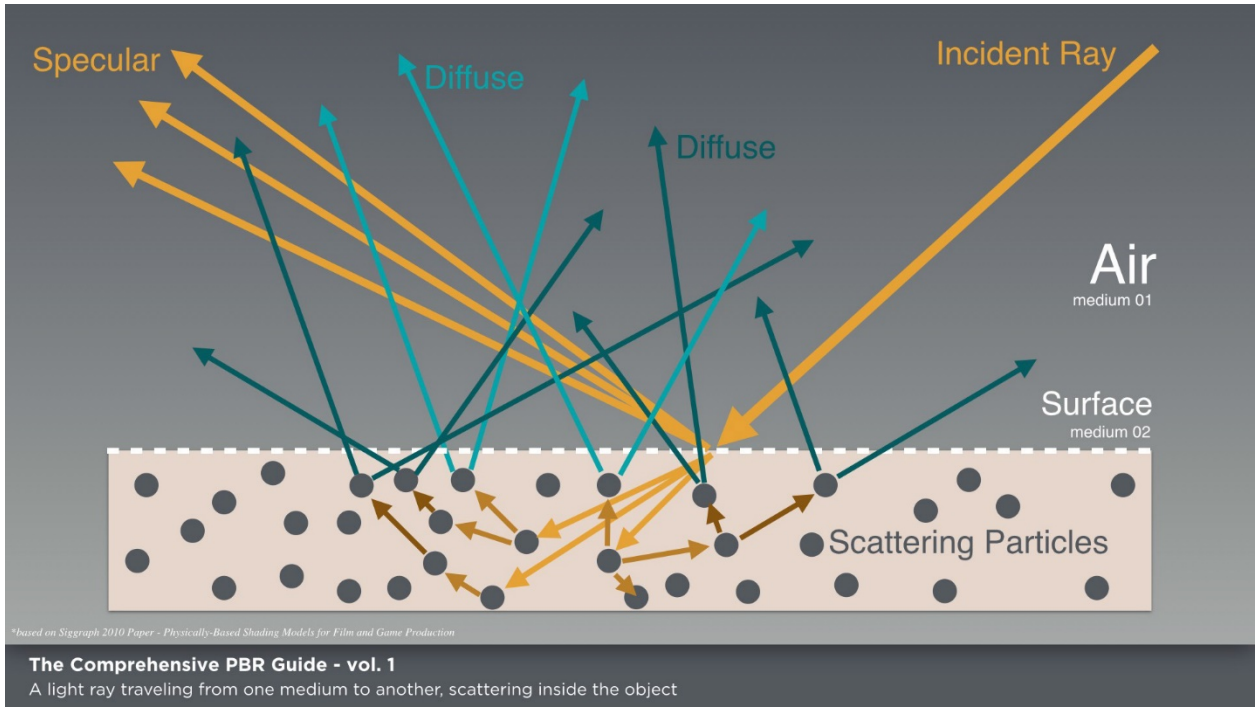




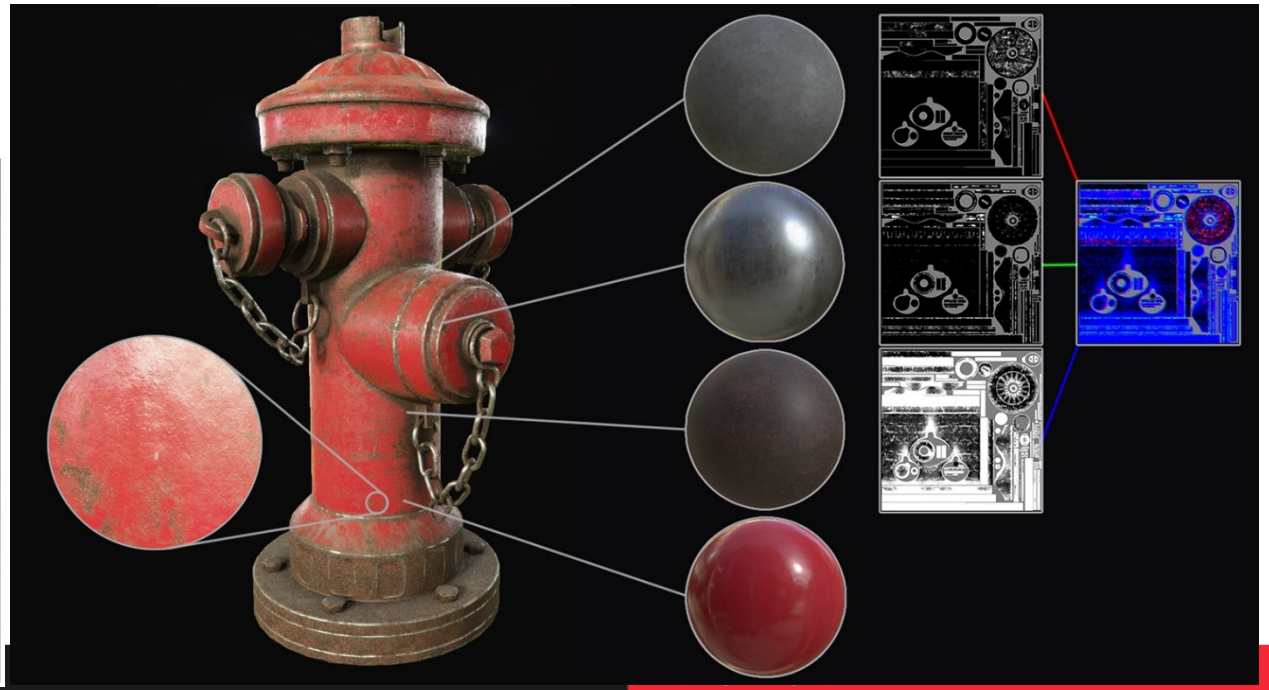
物体の定義.質感表現

- 図形を彩色することで
  様々な質感を表現
  - リアルに描く
    - 光学特性を正確に再現する
  - アニメに描く
    - 人間の視覚特性や感性を考慮する
    - 図面などに相応しい





The Comprehensive PBR Guide - vol. 1
A light ray traveling from one medium to another, scattering inside the object



▪OpenGL の歴史

・ＦＭＸ版：github.com/LUXOPHIA/OpenGL
・ＶＣＬ版：github.com/LUXOPHIA/OpenGL_VCL

README.md

歴史に学ぶ OpenGL

OpenGL のバージョンを辿りながら実装していくことで、新しい API が追加された意
造を把握していきます。

- OpenGL 1.0
- OpenGL 1.1
- OpenGL 1.5
- OpenGL 2.1
- OpenGL 3.0

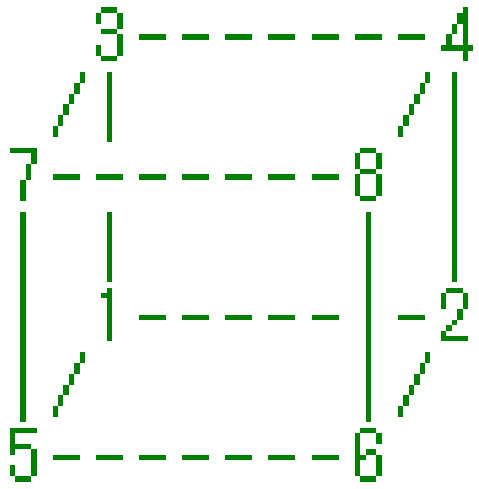
ここ見て！

OpenGL

FMX：FireMonkey フレームワークのコンポーネントとして OpenGL の描画領域を埋

OpenGL 1.0 (1992)

- glBegin ～ glEnd 関数で括る
- 頂点の属性毎に異なる関数
  - glVertex：位置
  - glColor：色
  - glNormal：法線
  - glTexCoord：テクスチャ座標




```
Ps :array [ 1..8 ] of TSingle3D
     = ( ( X:-1; Y:-1; Z:-1 ), ( X:+1; Y:-1; Z:-1 ),
         ( X:-1; Y:+1; Z:-1 ), ( X:+1; Y:+1; Z:-1 ),
         ( X:-1; Y:-1; Z:+1 ), ( X:+1; Y:-1; Z:+1 ),
         ( X:-1; Y:+1; Z:+1 ), ( X:+1; Y:+1; Z:+1 ) );
Cs :array [ 1..8 ] of TAlphaColorF
     = ( ( R:0; G:0; B:0; A:1 ), ( R:1; G:0; B:0; A:1 ),
         ( R:0; G:1; B:0; A:1 ), ( R:1; G:1; B:0; A:1 ),
         ( R:0; G:0; B:1; A:1 ), ( R:1; G:0; B:1; A:1 ),
         ( R:0; G:1; B:1; A:1 ), ( R:1; G:1; B:1; A:1 ) );
Es :array [ 1..6, 1..4 ] of Cardinal
     = ( ( 1, 5, 7, 3 ), ( 8, 6, 2, 4 ),
         ( 1, 2, 6, 5 ), ( 8, 4, 3, 7 ),
         ( 1, 3, 4, 2 ), ( 8, 7, 5, 6 ) );
```

```
glBegin( GL_QUADS );
  for N := 1 to 6 do
  begin
       for K := 1 to 4 do
       begin
            E := Es[ N, K ];
            with Cs[ E ] do glColor3f( R, G, B );
            with Ps[ E ] do glVertex3f( X, Y, Z );
       end;
  end;
glEnd;
```

1頂点ずつ送る

一気に描画

OpenGL 1.1 (1997)

- 頂点配列
  - 属性毎の配列を一気に転送可能

```
Ps :array [ 0..8-1 ] of TSingle3D
     = ( ( X:-1; Y:-1; Z:-1 ), ( X:+1; Y:-1; Z:-1 ),
         ( X:-1; Y:+1; Z:-1 ), ( X:+1; Y:+1; Z:-1 ),
         ( X:-1; Y:-1; Z:+1 ), ( X:+1; Y:-1; Z:+1 ),
         ( X:-1; Y:+1; Z:+1 ), ( X:+1; Y:+1; Z:+1 ) );
Cs :array [ 0..8-1 ] of TAlphaColorF
     = ( ( R:0; G:0; B:0; A:1 ), ( R:1; G:0; B:0; A:1 ),
         ( R:0; G:1; B:0; A:1 ), ( R:1; G:1; B:0; A:1 ),
         ( R:0; G:0; B:1; A:1 ), ( R:1; G:0; B:1; A:1 ),
         ( R:0; G:1; B:1; A:1 ), ( R:1; G:1; B:1; A:1 ) );
Es :array [ 0..12-1, 0..3-1 ] of Cardinal
     = ( ( 0, 4, 6 ), ( 6, 2, 0 ), ( 7, 5, 1 ), ( 1, 3, 7 ),
         ( 0, 1, 5 ), ( 5, 4, 0 ), ( 7, 3, 2 ), ( 2, 6, 7 ),
         ( 0, 2, 3 ), ( 3, 1, 0 ), ( 7, 6, 4 ), ( 4, 5, 7 ) );
```

位置配列を有効化

```
glEnableClientState( GL_VERTEX_ARRAY );
glEnableClientState( GL_COLOR_ARRAY  );
  glVertexPointer( 3, GL_FLOAT, 0, @Ps[ 0 ] );
  glColorPointer ( 4, GL_FLOAT, 0, @Cs[ 0 ] );
  glDrawElements( GL_TRIANGLES, 3{Poin} * 12{Face}, GL_UNSIGNED_INT, @Es[ 0, 0 ] );
glDisableClientState( GL_VERTEX_ARRAY );
glDisableClientState( GL_COLOR_ARRAY  );
```

色配列を有効化

位置配列を転送

色配列を転送

インデックス配列で描画

色配列を無効化

位置配列を無効化

OpenGL 1.5 (2003)

データ型　　更新頻度

```
_VerterP := TGLVerterS<TSingle3D>    .Create( GL_STATIC_DRAW );
_VerterC := TGLVerterS<TAlphaColorF>.Create( GL_STATIC_DRAW );
_Elemer  := TGLElemerFace32          .Create( GL_STATIC_DRAW );
```

- ＶＢＯ：Verttex Buffer Object
  - 予めＧＰＵ側へ頂点配列を転送し保持可能
    - 描画速度の大幅な向上

頂点配列を転送

```
_VerterP.Import( Ps );
_VerterC.Import( Cs );
_Elemer .Import( Es );
```

```
glEnableClientState( GL_VERTEX_ARRAY );
glEnableClientState( GL_COLOR_ARRAY  );

  with _VerterP do
  begin
       Bind;
         glVertexPointer( 3, GL_FLOAT, 0, nil );
       Unbind;
  end;

  with _VerterC do
  begin
       Bind;
         glColorPointer( 4, GL_FLOAT, 0, nil );
       Unbind;
  end;

  with _Elemer do
  begin
       Bind;
         glDrawElements( GL_TRIANGLES, 3{Poin} * 12{Face}, GL_UNSIGNED_INT, nil );
       Unbind;
  end;

glDisableClientState( GL_VERTEX_ARRAY );
glDisableClientState( GL_COLOR_ARRAY  );
```

バインド　転送不要　利用登録

バインド　転送不要　利用登録

バインド　転送不要　描画

OpenGL 1.5. V B O

- 生成/使用方法は皆同じ
  - 種類フラグが異なるだけ

```
constructor TGLBuffer<_TYPE_>.Create( const Usage_:GLenum );
begin
      inherited Create;

      _Align := InitAlign;
      _Strid := InitStrid;

      glGenBuffers( 1, @_ID );

      _Usage := Usage_;
       Count := 0;
end;

destructor TGLBuffer<_TYPE_>.Destroy;
begin
      glDeleteBuffers( 1, @_ID );

      inherited;
end;
```

生成

廃棄

ＶＢＯの有効化

```
procedure TGLBuffer<_TYPE_>.Bind;
begin
      glBindBuffer( GetKind, _ID );
end;

procedure TGLBuffer<_TYPE_>.Unbind;
begin
      glBindBuffer( GetKind, 0 );
end;
```

ＶＢＯの無効化

```
procedure TGLBuffer<_TYPE_>.Import( const Array_:array of _TYPE_ );
begin
      _Count := Length( Array_ );

      Bind;

        glBufferData( GetKind, SizeOf( Array_ ), @Array_[ 0 ], _Usage );

      Unbind;
end;
```

頂点配列の転送

種類フラグ

OpenGL 2.1 (2006)

- プログラマブルシェーダーが実現
  - ＳＯ：シェーダーオブジェクト
    - 質感を計算するプログラム
  - ＰＯ：プログラムオブジェクト
    - ＳＯをまとめる実行主体

```
glEnableClientState( GL_VERTEX_ARRAY );
glEnableClientState( GL_COLOR_ARRAY  );

  with _VerterP do
  begin
       Bind;
         glVertexPointer( 3, GL_FLOAT, 0, nil );
       Unbind;
  end;

  with _VerterC do
  begin
       Bind;
         glColorPointer( 4, GL_FLOAT, 0, nil );
       Unbind;
  end;

  with _Elemer do
  begin
       Bind;

         with _Progra do
         begin
              Use;

                glDrawElements( GL_TRIANGLES, 3{Poin} * 12

              Unuse;
         end;

       Unbind;
  end;

glDisableClientState( GL_VERTEX_ARRAY );
glDisableClientState( GL_COLOR_ARRAY  );
```

```
with _Elemer do
begin
     Bind;

       with _Progra do
       begin
            Use;

              glDrawElements( GL_TRIANGLES, 3{Poin} * 12{Face}, GL_UNSIGNED_INT, nil );

            Unuse;
       end;

     Unbind;
end;
```

ＰＯのバインド

形状の描画

OpenGL 2.1.ＧＬＳＬ

- ＧＬＳＬ言語でシェーダを記述
  - ＶＳ：バーテックスシェーダー
    - 頂点毎に呼ばれるイベント
  - ＦＳ：フラグメントシェーダー
    - ピクセル毎に呼ばれるイベント

```
with _Progra do
begin
      Attach( _ShaderV );
      Attach( _ShaderF );

      Link;
end;
```

ＶＳの登録

ＦＳの登録

リンク

```
with _ShaderF.Source do
begin
      BeginUpdate;
        Add( '#version 120' );
        Add( 'void main()' );
        Add( '{' );
        Add( '  gl_FragColor = gl_Color;' );
        Add( '}' );
      EndUpdate;
end;
```

ＦＳのソース

```
with _ShaderV.Source do
begin
      BeginUpdate;
        Add( '#version 120' );
        Add( 'void main()' );
        Add( '{' );
        Add( '  gl_Position   = gl_ModelViewProjectionMatrix * gl_Vertex;' );
        Add( '  gl_FrontColor = gl_Color;' );
        Add( '}' );
      EndUpdate;
end;
```

ＶＳのソース

OpenGL 3.0（2008）

- ＶＡＯ：Vertex Array Object
  - ＶＢＯの**有効/無効状態**をまとめて登録しておく辞書
    - 描画の度に複数のＶＢＯをバインドする手間が省ける
- Core Profile 利用時はＶＡＯの利用が必須

```
with _Progra do
begin
      Use;

      with _Varray do
      begin
            Bind;

              glDrawElements( GL_TRIANGLES, 3{Poin} * 12{Face}, GL_UNSIGNED_INT, nil );

            Unbind;
      end;

      Unuse;
end;
```

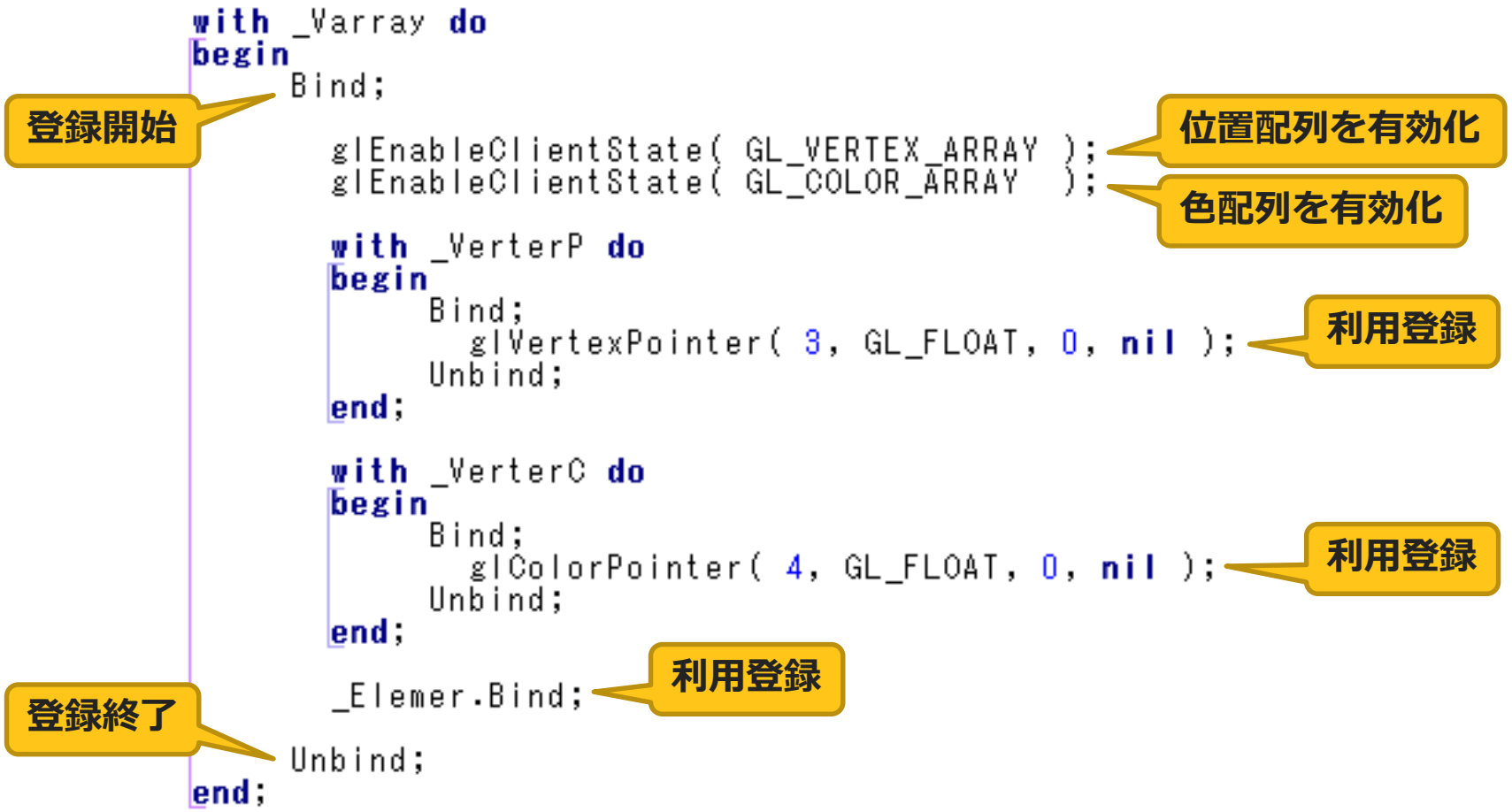
ＰＯを有効化

ＶＡＯを有効化

形状の描画

OpenGL 3.0. V A O

- 予め利用するＶＢＯを登録する



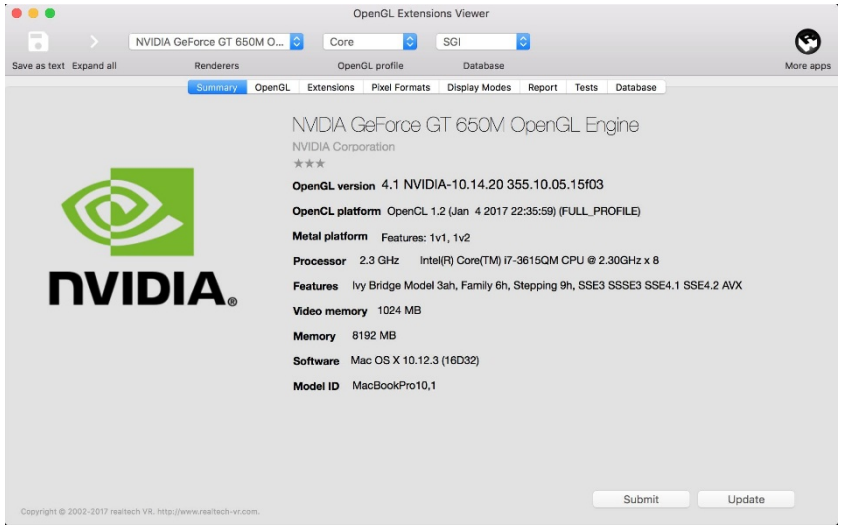
Post OpenGL 3.0

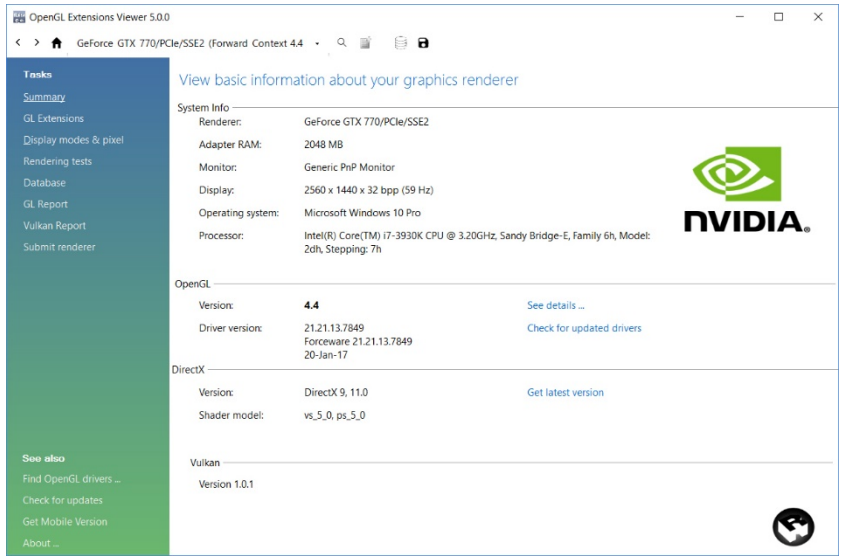
- OpenGL 3.0
  - シェーダで代替できる古い関数は非推奨にするよ
    - www.khronos.org/opengl/wiki/Legacy_OpenGL
- OpenGL 3.1
  - 非推奨な関数はすべて削除（拡張機能扱い）したよ
- OpenGL 3.2
  - 互換モードを選べば使えるようにしたよ
    - **Core Profile**：モダン関数のみモード
    - **Compatible Profile**：レガシー関数含有モード

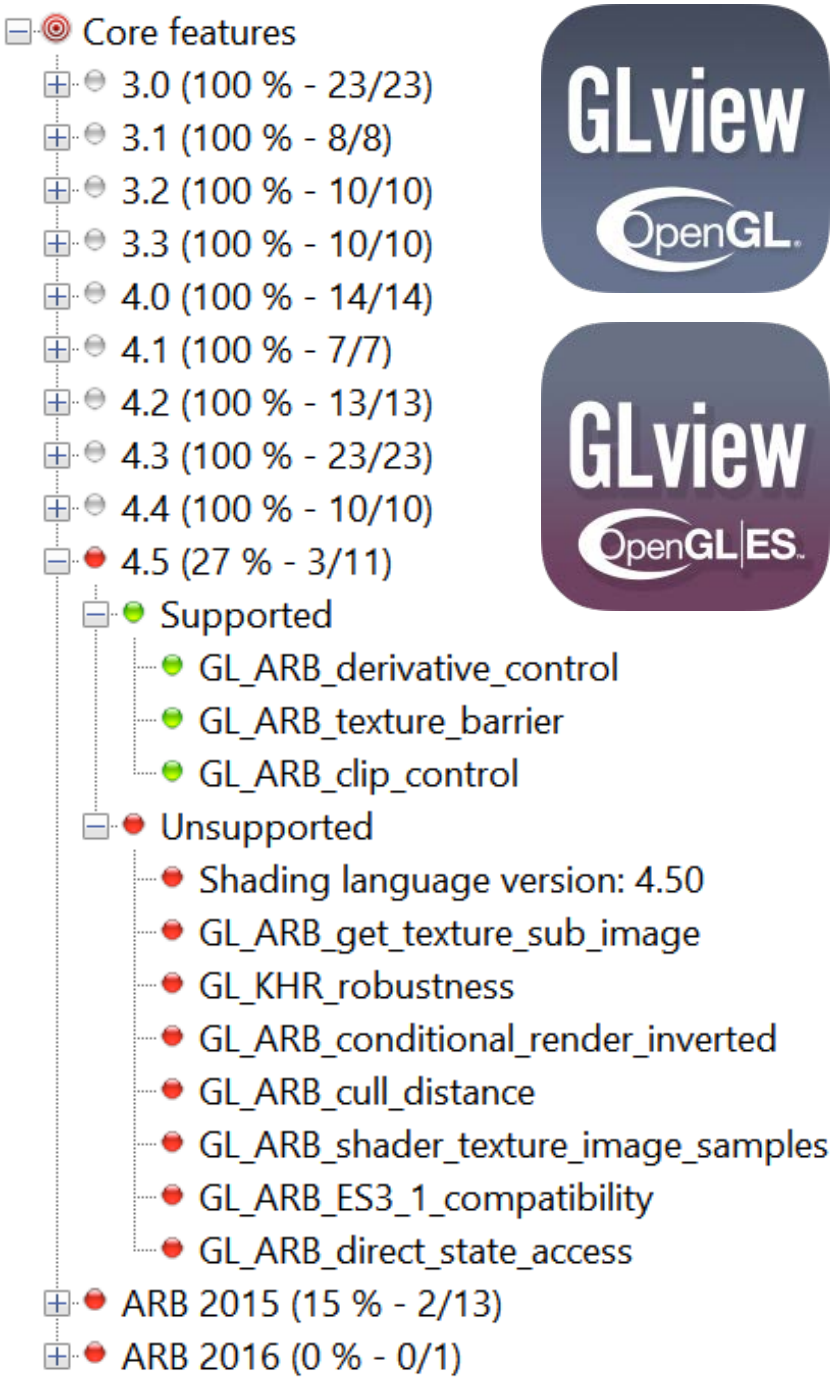
  ～ ～ ～ ～ ～
- OpenGL 4.3
  - モダンな機能がほぼ出そろう
    - 今から勉強するならここから！

OpenGL Extensions Viewer

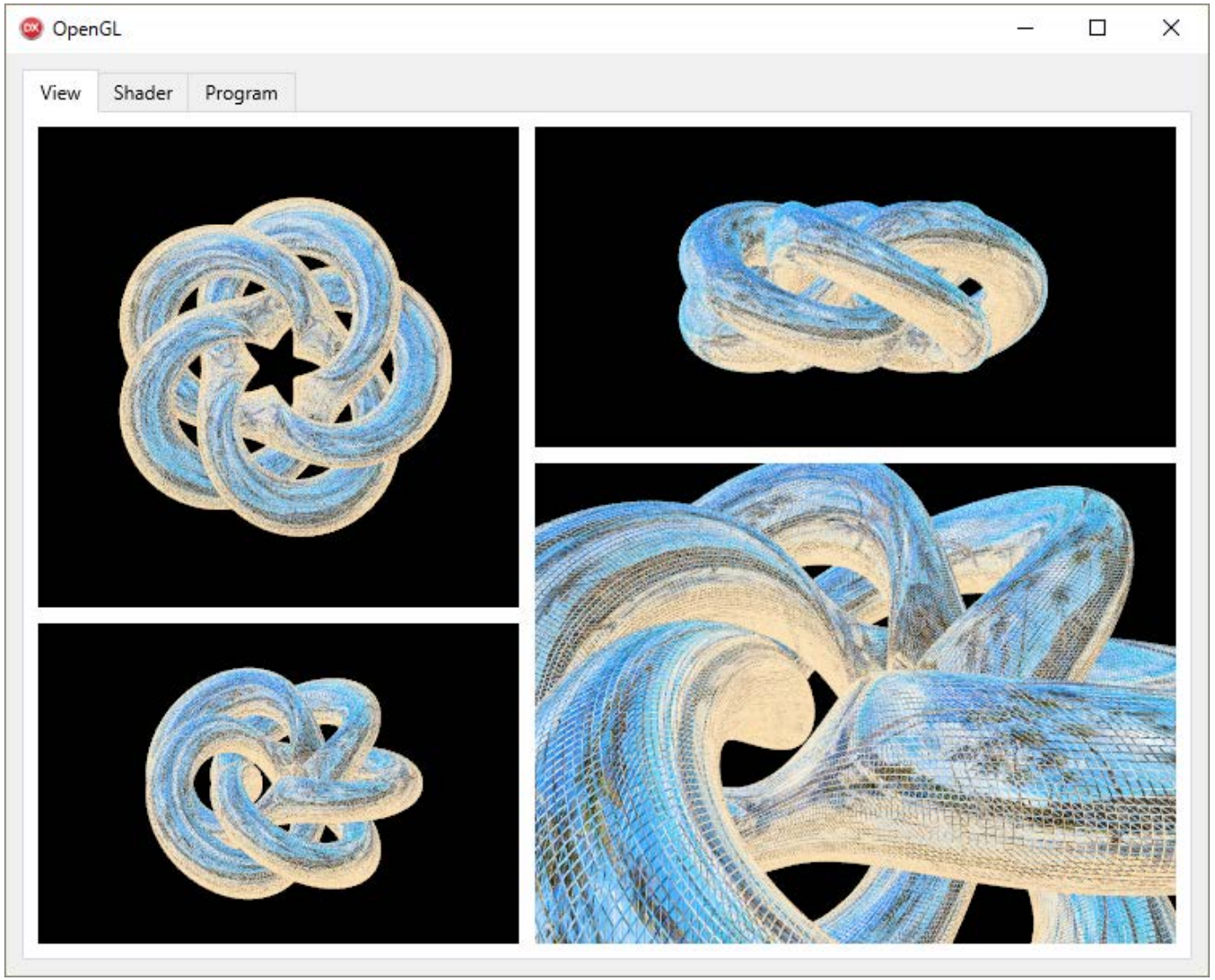
- realtech-vr.com/admin/glview
- OpenGL のバージョンを調べられるアプリ
  - 各種ＡＰＩの実装率も分かる
- マルチプラットフォーム
  - Windows, macOS, Android, iOS

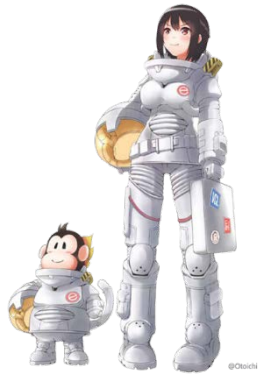






■モダンな実装





embarcadero DEVELOPER CAMP

モダンな実装.座標変換

- 従来の行列系関数はすべて使えない
  - すべて自前で計算する
- 頂点の座標変換はＶＳで代替する

```
GLViewer1.OnPaint := procedure
begin
      glMatrixMode( GL_PROJECTION );
        glLoadIdentity;
        glOrtho( -2, +2, -2, +2, _N, _F );
      glMatrixMode( GL_MODELVIEW );
        glLoadIdentity;
        glTranslatef( 0, 0, -5 );
        glRotatef( +90, 1, 0, 0 );
        glRotatef( _Angle, 0, 1, 0 );
        DrawShaper;
end;
```

```
gl_Position = _ViewerScal
            * _Camera.Proj
            * inverse( _Camera.Pose )
            * _Shaper.Pose
            * _VerterPos;
```

スクリーン座標

ウィンドウ比率

カメラの投影行列

逆行列

カメラの姿勢行列

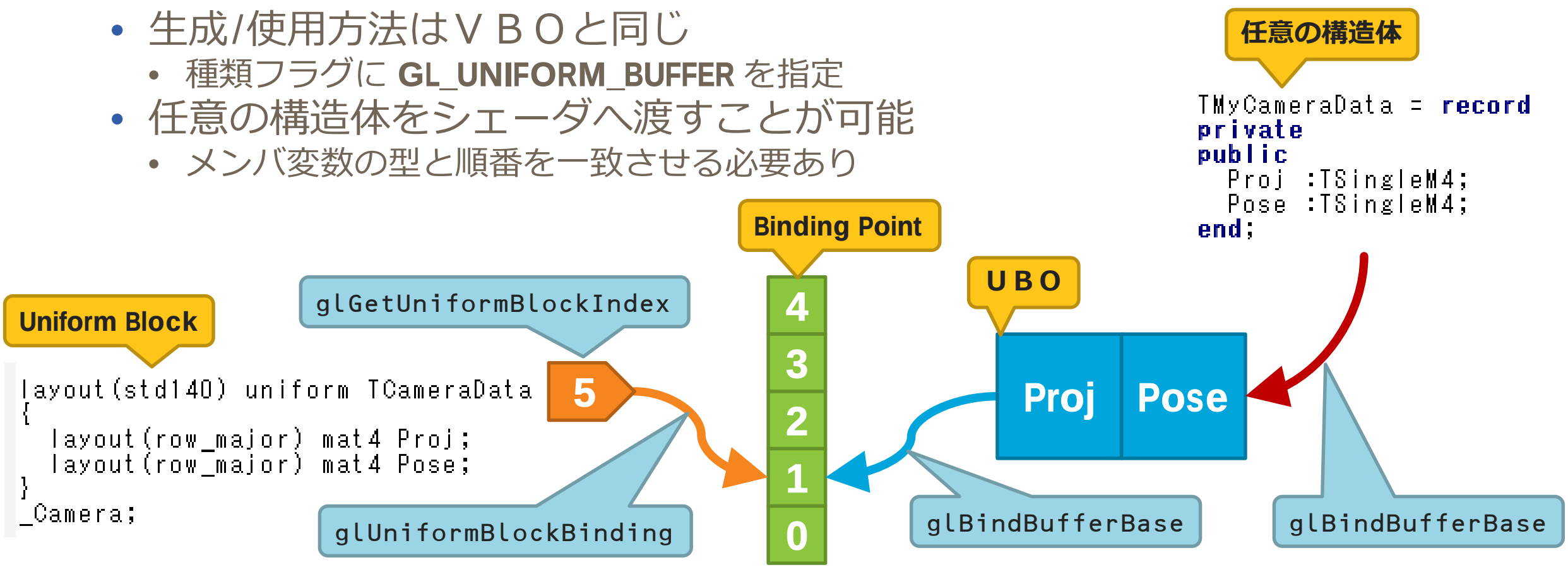
物体の姿勢行列

頂点座標

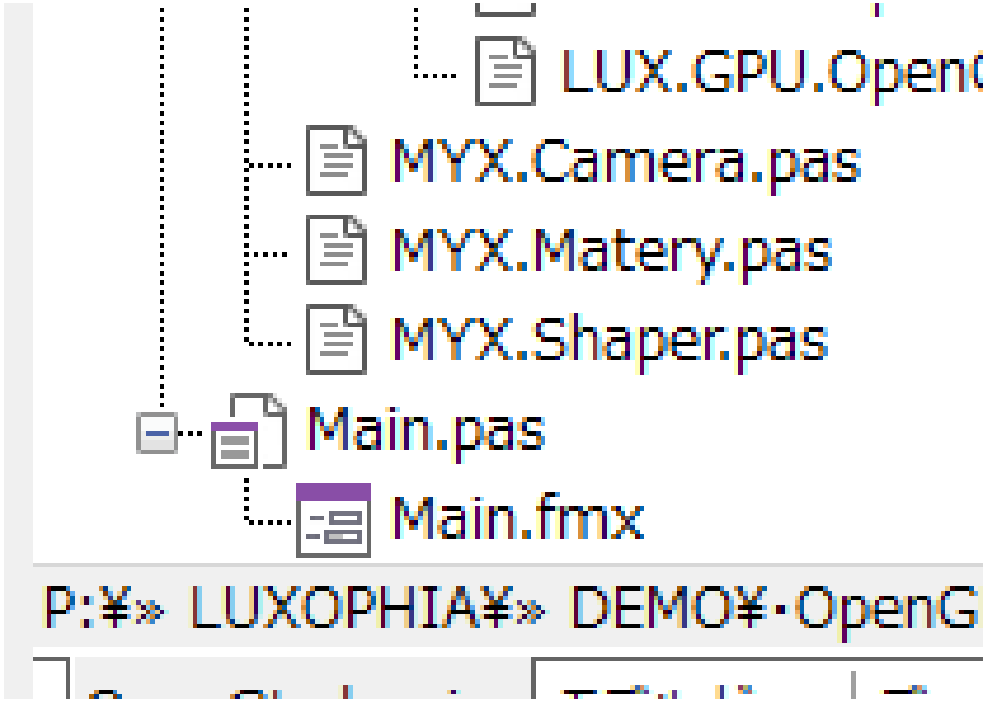
- 物体の姿勢行列は変数としてシェーダに読み込ませる

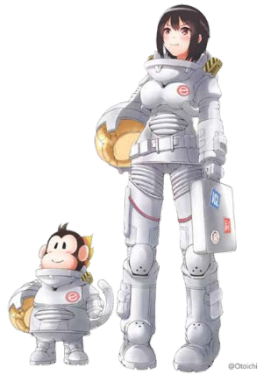
モダンな実装.ＵＢＯ

- **ＵＢＯ**：Unoform Buffer Object
  - すべてのシェーダから共通の定数として参照できる配列
  - 生成/使用方法はＶＢＯと同じ
    - 種類フラグに **GL_UNIFORM_BUFFER** を指定
  - 任意の構造体をシェーダへ渡すことが可能
    - メンバ変数の型と順番を一致させる必要あり



- ミニマムライブラリ





embarcadero DEVELOPER CAMP

ミニマムライブラリ.クラス構造

- ３つのクラスに整理
  - カメラ
    - **TMyCamera**
  - 材質（質感）
    - **TMyMatery**
  - 物体（形状）
    - **TMyShaper**
- 描画は３行

```
GLViewer1.OnPaint := procedure
begin
      _Camera1.Use;
      _Matery .Use;
      _Shaper .Draw;
end;
```

カメラを有効化

材質を有効化

物体を描画

```
with C do
begin
      Proj := TSingleM4.ProjPers( -_N/2, +_N/2, -_N/2, +_N/2, _N, _F );

      Pose := TSingleM4.RotateX( DegToRad( -45 ) )
            * TSingleM4.Translate( 0, 0, +2 );
end;

_Camera4.Data := C;
```

投影行列を設定

カメラの
姿勢行列を設定

カメラのＵＢＯへ入力

```
with _Matery do
begin
      ShaderV.Source.LoadFromFile( '..¥..¥_DATA¥ShaderV.glsl' );

      ShaderF.Source.LoadFromFile( '..¥..¥_DATA¥ShaderF.glsl' );

      Imager.LoadFromFile( '..¥..¥_DATA¥Spherical_1024x1024.png' );
end;
```

シェーダソースのロード

テクスチャのロード

```
with _Shaper do
begin
      LoadFormFunc( BraidedTorus, 1300, 100 );

      with S do
      begin
            Pose := TSingleM4.Identify;
      end;

      Data := S;
end;
```

ポリゴン生成

物体の姿勢行列を設定

物体のＵＢＯへ入力

ミニマムライブラリ. TMyCamera

- 投影行列と姿勢行列をＵＢＯで保持

```
TMyCameraData = record
private
public
  Proj :TSingleM4;
  Pose :TSingleM4;
end;

TMyCamera = class
private
protected
  _Data :TGLUnifor<TMyCameraData>;
  ///// アクセス
  function GetData :TMyCameraData;
  procedure SetData( const Data_:TMyCameraData );
public
  constructor Create;
  destructor Destroy; override;
  ///// プロパティ
  property Data :TMyCameraData read GetData write SetData;
  ///// メソッド
  procedure Use;
end;
```

ＵＢＯ構造体

投影行列

姿勢行列

ＵＢＯ

```
function TMyCamera.GetData :TMyCameraData;
begin
      Result := _Data[ 0 ];
end;

procedure TMyCamera.SetData( const Data_:TMyCameraData );
begin
      _Data[ 0 ] := Data_;
end;
```

ＵＢＯへの入出力

```
procedure TMyCamera.Use;
begin
      _Data.Use( 1{BinP} );
end;
```

有効化

ミニマムライブラリ. TMyMatery

- シェーダー関連オブジェクトを内包
- テクスチャオブジェクトを保持

```
TMyMatery = class
private
protected
  _ShaderV :TGLShaderV;
  _ShaderF :TGLShaderF;
  _Engine  :TGLEngine;
  _Sample  :TGLSample;
  _Imager  :TGLImager2D_RGBA;
public
  constructor Create;
  destructor Destroy; override;
  //// プロパティ
  property ShaderV :TGLShaderV        read _ShaderV;
  property ShaderF :TGLShaderF        read _ShaderF;
  property Engine  :TGLEngine         read _Engine ;
  property Sample  :TGLSample         read _Sample ;
  property Imager  :TGLImager2D_RGBA  read _Imager ;
  //// メソッド
  procedure Use;
end;
```

頂点ＳＯ

断片ＳＯ

ＰＯ

サンプラーオブジェクト

テクスチャオブジェクト

有効化

```
procedure TMyMatery.Use;
begin
      _Engine.Use;

      _Sample.Use( 0{BinP} );
      _Imager.Use( 0{BinP} );
end;
```

ＰＯ有効化

テクスチャ有効化

ミニマムライブラリ. TMyMatery.Create

- 変数⇒ＢＰの紐付けを自動化
  - Verters：ＶＢＯ用
    - 新しいＡＰＩではＶＢＯも
      ＵＢＯのようにＢＰ経由で接続可能
  - Unifors：ＵＢＯ用
  - Imagers：テクスチャ用
    - ＵＢＯのようにテクスチャユニット
      という単位で接続可能
  - Framers：フレーム用
    - ウィンドウへだけでなく
      内部的に確保した複数の画像へ
      レンダリングすることも可能

```
constructor TMyMatery.Create;
begin
     inherited;

     _ShaderV := TGLShaderV       .Create;
     _ShaderF := TGLShaderF       .Create;
     _Engine  := TGLEngine        .Create;
     _Sample  := TGLSample        .Create;
     _Imager  := TGLImager2D_RGBA.Create;

     with _Engine do
     begin
          Attach( _ShaderV{Shad} );
          Attach( _ShaderF{Shad} );

          with Verters do
          begin
               Add( 0{BinP}, '_VerterPos'{Name}, 3{EleN}, GL_FLOAT{EleT} );
               Add( 1{BinP}, '_VerterNor'{Name}, 3{EleN}, GL_FLOAT{EleT} );
               Add( 2{BinP}, '_VerterTex'{Name}, 2{EleN}, GL_FLOAT{EleT} );
          end;

          with Unifors do
          begin
               Add( 0{BinP}, 'TViewerScal'{Name} );
               Add( 1{BinP}, 'TCameraData'{Name} );
               Add( 2{BinP}, 'TShaperData'{Name} );
          end;

          with Imagers do
          begin
               Add( 0{BinP}, '_Imager'{Name} );
          end;

          with Framers do
          begin
               Add( 0{BinP}, '_FramerCol'{Name} );
          end;
     end;
end;
```

ＰＯへのＳＯ登録

変数名⇒ＢＰの登録

ミニマムライブラリ. TMyShaperBase

- TMyShaper の基底クラス
  - 姿勢行列をＵＢＯで保持

```
TMyShaperData = record
private
public
  Pose :TSingleM4;
end;

TMyShaperBase = class
private
protected
  _Data :TGLUnifor<TMyShaperData>;
  ///// アクセス
  function GetData :TMyShaperData;
  procedure SetData( const Data_:TMyShaperData );
public
  constructor Create;
  destructor Destroy; override;
  ///// プロパティ
  property Data :TMyShaperData read GetData write SetData;
  ///// メソッド
  procedure Draw; virtual;
end;
```

ＵＢＯ構造体

姿勢行列

ＵＢＯ

ＵＢＯへの入出力

描画

ミニマムライブラリ. TMyShaper

- 各種ＶＢＯを内包

```
TMyShaper = class( TMyShaperBase )
private
protected
  _PosBuf :TGLVerterS<TSingle3D>;
  _NorBuf :TGLVerterS<TSingle3D>;
  _TexBuf :TGLVerterS<TSingle2D>;
  _EleBuf :TGLElemerFace32;
public
  constructor Create;
  destructor Destroy; override;
  //// プロパティ
  property PosBuf :TGLVerterS<TSingle3D> read _PosBuf;
  property NorBuf :TGLVerterS<TSingle3D> read _NorBuf;
  property TexBuf :TGLVerterS<TSingle2D> read _TexBuf;
  property EleBuf :TGLElemerFace32       read _EleBuf;
  //// メソッド
  procedure Draw; override;
  procedure LoadFormFunc( const Func_:TConstFunc<TdSingle2D,TdSingle3D>; const DivX_,DivY_:Integer );
end;
```

位置ＶＢＯ

法線ＶＢＯ

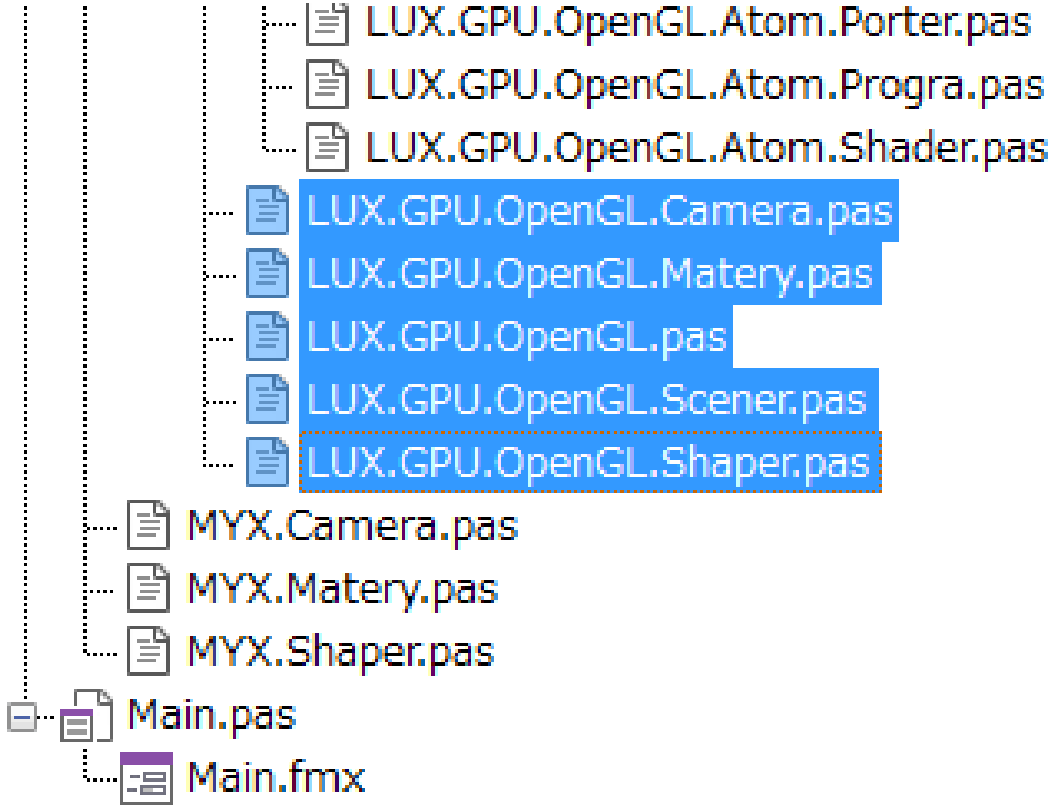
テクスチャ座標ＶＢＯ

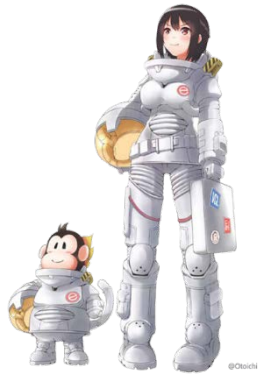
インデックスＶＢＯ

描画

形状のロード

- マキシマムライブラリ

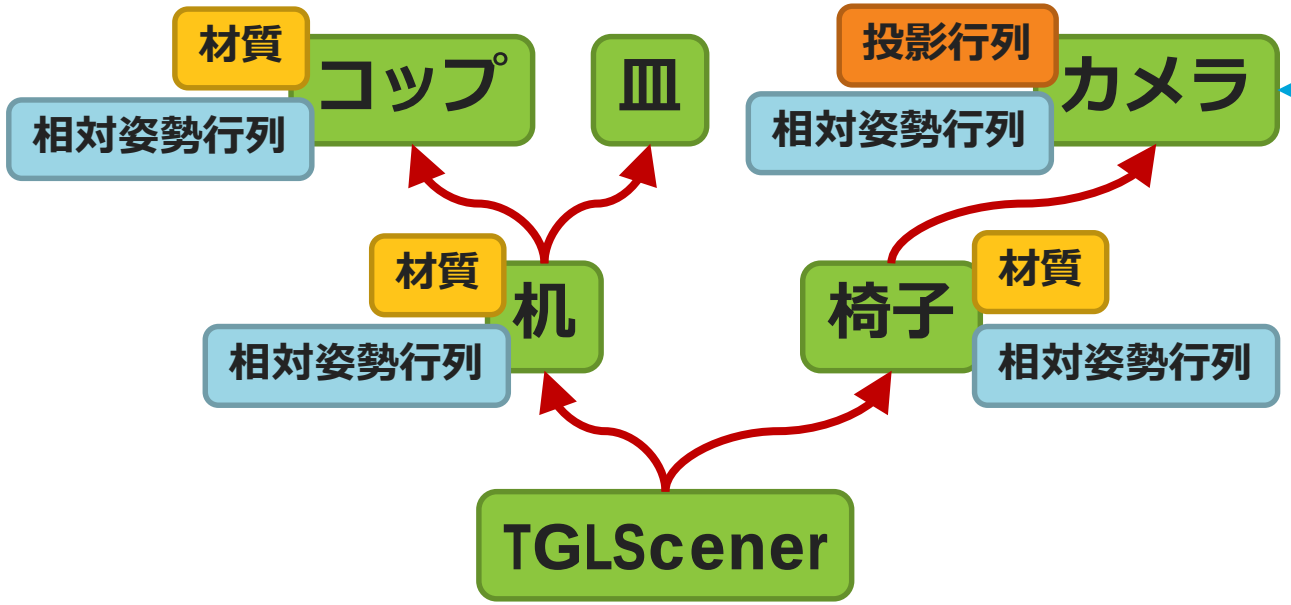


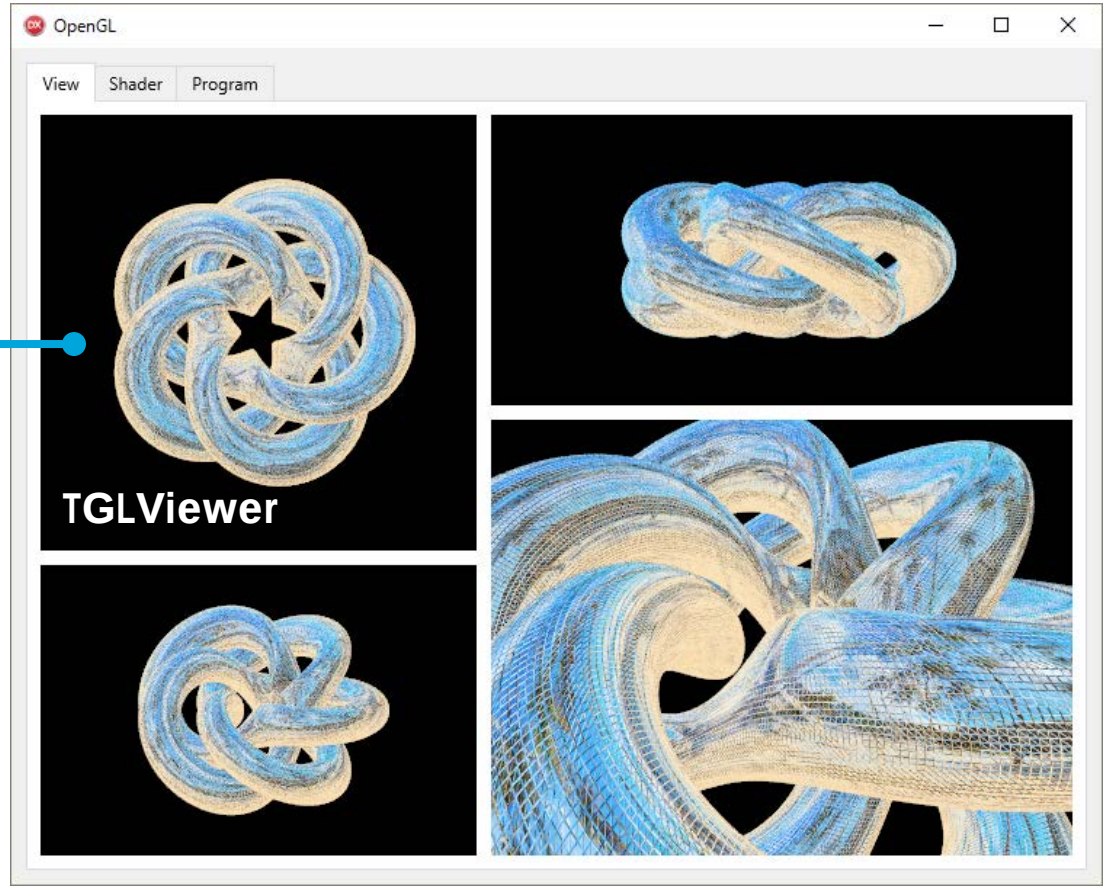


embarcadero DEVELOPER CAMP

マキシマムライブラリ

- ツリー構造でシーン構築できるノードベースライブラリ
- ジオメトリシェーダに対応
- TGLViewer をカメラに紐付け
  - Repaint メソッドによる直感的な描画





THANKS!

www.embarcadero.com/jp

第34回 エンバカデロ・デベロッパーキャンプ