WebGL によるデータ可視化入門*

はじめに

陰山 聡

神戸大学 システム情報学研究科 計算科学専攻

2014.04.15

*情報可視化論 X021（2014 年前期） 自然 3 号館 1F 演習室

講義の目標：WebGL とは

OpenGL の歴史

シェーダとシェーディング言語： GLSL

コンピュータグラフィックス

はじめに

講義のウェブページ

http://bit.ly/1qXgljB

- 各種連絡
- 講義資料
- 作品掲載

調査：演習室端末の id

- if 計算科学演習 I を履修 → id は今週発行される
- else → 新たに発行するので学籍番号と氏名（漢字、ひらがな、ローマ字） をメールで。今日中に。
- メールアドレス kageyama.lecture

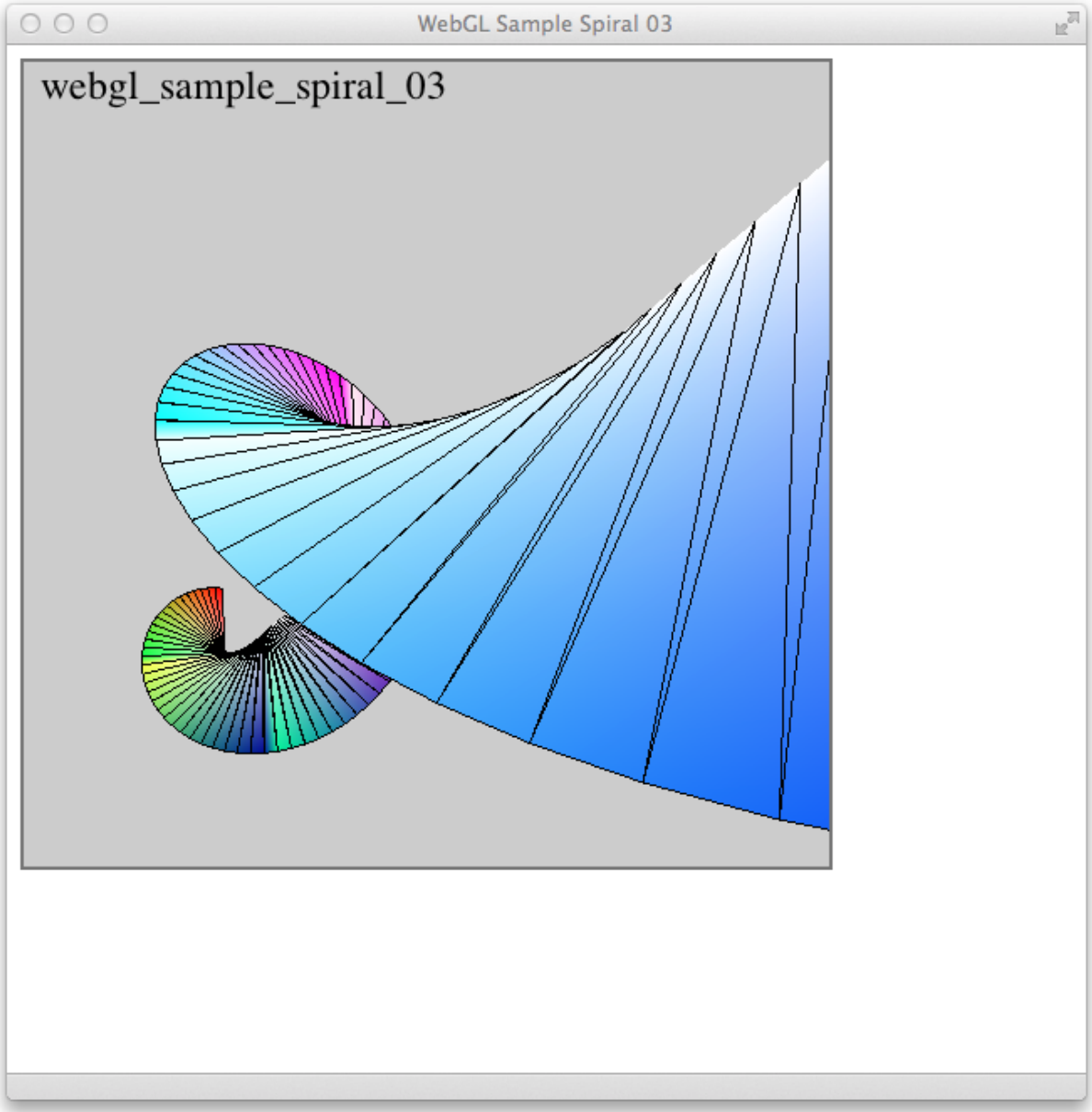
予備知識

- “ 古い ” OpenGL
  - 実験・プロジェクト演習で経験しているはず
  - glBegin/glEnd, glVertex, glNormal は覚えている？
- CG（Computer Graphics）の基礎
  - 正射影、透視射影は知っているか？
  - OpenGL の照明モデルは知っているか？
- 線形代数
  - 今回（次回）復習

目標

- WebGL を使って簡単なアプリケーションが作れるようになる
- シェーダ言語を使えるようになる
- 科学データの可視化手法を理解する

目標



成績

レポート

- 計 5 回（20 点 x 5 = 100 点）（の予定）
- メールで提出。アドレス：kageyama.lecture
- 一部の作品は講義のウェブページに掲載し、作者名も記す。
- 自分の名前（名字）を掲載されたくない場合は希望仮名（イニシャル）をレポートに明記すること。

WebGL とは

WebGL - OpenGL ES 2.0 for the Web[†]

[†] http://www.khronos.org/webgl/

WebGL 対応ブラウザ

- Firefox 4
- Google Chrome 8
- Safari 5
- Opera 12
- Internet Explorer 11

ブラウザとバージョンによっては設定が必要かもしれない。デモ[‡]が見えるかどうかチェックしておくこと。

[‡] http://www.khronos.org/webgl/wiki/Demo_Repository

参考書

Professional WebGL Programming, A. Anyuru

実践プログラミング WebGL, 吉川郁夫訳

Data Visualization Principles and Practice, C. Telea

three.js による HTML5 3D グラフィックス, 上下, 遠藤理平

講義計画

- Part 1
  - OpenGL の歴史
  - WebGL とは
  - CG と GPU の基礎
  - 数学的準備
- Part 2
  - WebGL を通じたシェーダ入門
- Part 3
  - 様々な可視化手法の解説

登攀ルート


GPUを使いこなしたSci. Viz.
基本的可視化
アルゴリズム
シェーダ言語
コンピュータグラフィクス
線形代数

OpenGL の歴史

OpenGL とは

OpenGL 2.X は、OpenGL 1.X の機能を全て含む。

OpenGL 3.0 で上位互換性放棄

シェーダで置き換え可能な機能
OpenGL 3.0 非推奨機能
→ OpenGL 3.1 拡張機能
→ OpenGL 3.2 互換プロファイル

OpenGL Overview

`http://www.opengl.org`

OpenGL was first created as an open and reproducable alternative to Iris GL ...Silicon Graphics workstations... OpenGL 1.0 ...non-SGI 3rd party...

...

OpenGL 2.0...OpenGL Shading Language (also called GLSL), a C like language with which the transformation and fragment shading stages of the pipeline can be programmed.

OpenGL 3.0 adds the concept of deprecation[§]... GL 3.1 removed most deprecated features, and GL 3.2 created the notion of core and compatibility OpenGL contexts.

Official versions of OpenGL released to date are 1.0, 1.1, 1.2, 1.2.1, 1.3, 1.4, 1.5, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3, 4.0, 4.1, 4.2, 4.3.

[§]非推奨

OpenGL 関係

`http://www.khronos.org`

- OpenGL 4.3
- OpenGL SL 4.3 （シェーディング言語）
- OpenGL ES 3.0 組み込みシステム用） iOS, Android, Symbian
- WebGL 2.0

Shader-based OpenGL による OpenGL 教育

OpenGL はグラフィックスの API として長年教育現場で教えられてきた。
仕様の変化も少なかったので、長年同じ教科書、ノートが使われてきた。
ところが OpenGL 3.0 から、OpenGL は大きく変化（Shader-base 化）。
それに伴って教育内容も大改訂が必要となった。

OpenGL ver. 3.1 からの大きな変化

全てのアプリケーションは少なくとも一つの頂点シェーダと、少なくとも一つのフラグメントシェーダを与える必要がある。

OpenGL ver. 3.1 からの大きな変化

```
//
//  Hello_World.c
//
// To compile on Mac.
// gcc this_source.c -framework GLUT -framework OpenGL
//

#include <GLUT/glut.h>

void display (void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_POLYGON);
   glVertex2f(-0.5, -0.5);
   glVertex2f(-0.5,  0.5);
```

```
    glVertex2f( 0.5,  0.5);
    glVertex2f( 0.5, -0.5);
    glEnd();
    glutSwapBuffers();
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);
    glutCreateWindow("Hello World");
    glutDisplayFunc(display);
    glutMainLoop();
}
```

Listing 1: 古い OpenGL コード

WebGL（シェーダ）による三角形の描画

→ link: webgl_sample_triangle_00.html

はじめに

OpenGL シェーディング言語（OpenGL SL, GLSL）4.0

GPU を使うための言語

CG ソースコード	=	OpenGL ソースコード
	+	GLSL(フラグメントシェーダ）ソースコード
	+	GLSL(頂点シェーダ）ソースコード

OpenGL シェーディング言語

GPU 内部で走る単体のプログラム

専用言語

シェーディングプログラム（あるいは単にシェーダ）と呼ばれる

要するに数値演算プログラムの一種

シェーディングに限らず様々な数値処理が可能（→ GPGPU）

並列処理

GPU のプロセッサ数

GeForce GTX TITAN

シェーダプロセッサ “CUDA Cores” 2688 個

以前は**固定機能パイプライン**

現在はプログラマブル

固定機能パイプラインは廃止予定

グラフィックス

- immediate-mode
  - 即時モード API
  - 描画のたびにシーン全体を GPU に送る。たとえ変更がなくても。
- retained-mode
  - 保持モード API
  - シーン全体をまず GPU に送る。その後は変更部分だけを送る。

WebGL は即時モード API

OpenGL プロファイル

OpenGL 3.0

deprecation（非推奨）を導入 → 不要な関数を段階的に取り除く

OpenGL 3.2 より

コア プロファイルと互換プロファイル

プロファイルはウィンドウシステム API で選択

glBegin/glEnd は非推奨

標準的な行列 GL_MODELVIEW, GL_PROJECTION もコアプロファイルから削除。

OpenGL ES 2.0

OpenGL for Embedded Systems （組み込みシステム用）

Immediate モード API

glBegin/glEnd なし。頂点配列を使う。

シェーダベース。シェーダプログラムが必要。

アプリケーションプログラム： C/C++, Objective-C, Java

OpenGL ES 2.0 ≈ WebGL

WebGL は JavaScript で呼ぶ。

HTML5 canvas

HTML5： 第 5 世代の HTML

canvas: JavaScript で描画できる長方形領域

→ 【HTML5 canvas サンプルプログラム】

コンピュータグラフィックス（CG）

CG と並列計算

並列計算については学んできたはず。

空間を分割して、通信用のメモリを確保して、MPI で通信して ···

並列プログラム作りは結構大変。← 結局同期が大変だから。

一方、同期について気にしなくてもよい問題もたくさんある。もともとのアルゴリズムが並列化可能なもの。→ “Embarrassingly” parallel problems

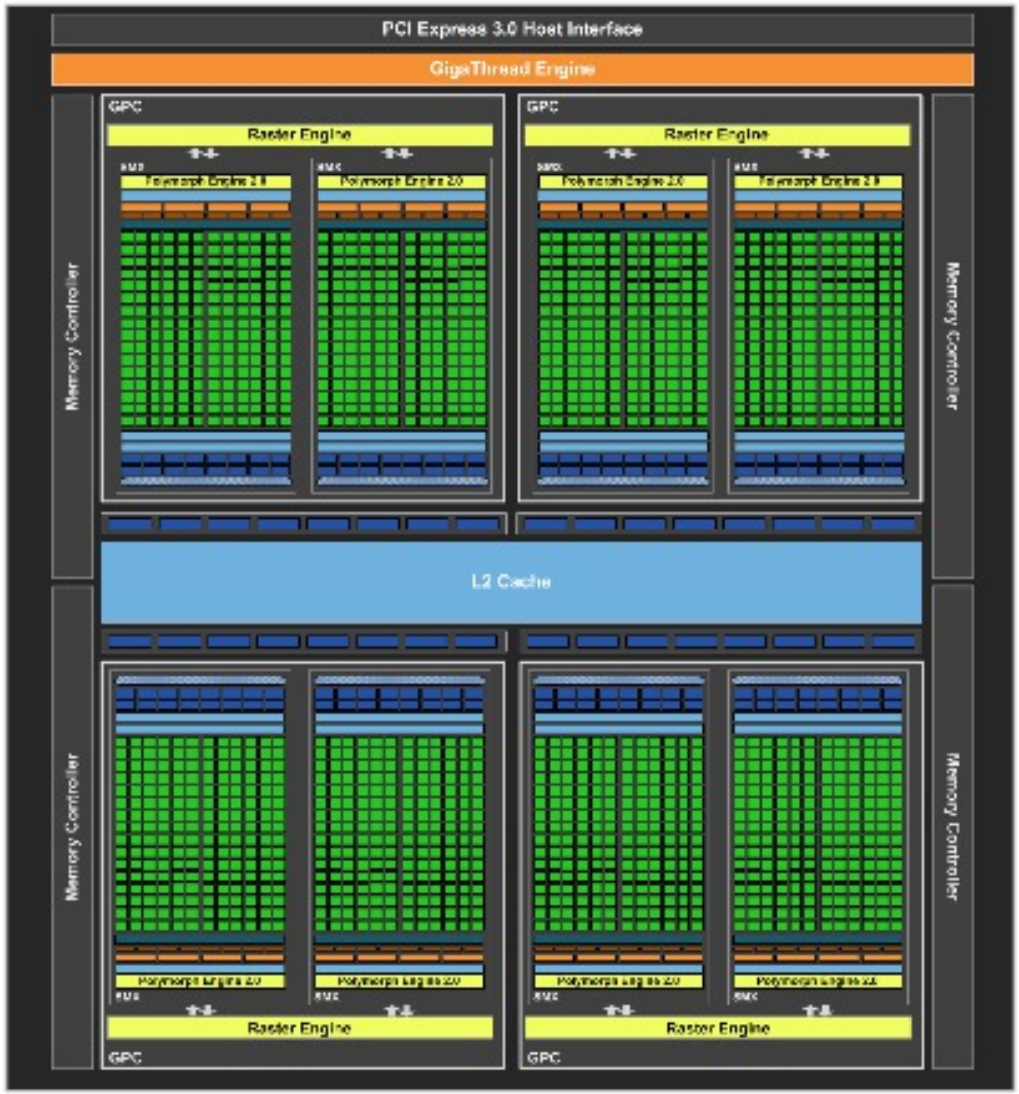
CG の処理の多くが embarrassingly parallel

座標変換、テクスチャマップ、シェーディング、ラスタ化、etc.

CG 専用並列計算機 → GPU (Graphics Processing Unit)

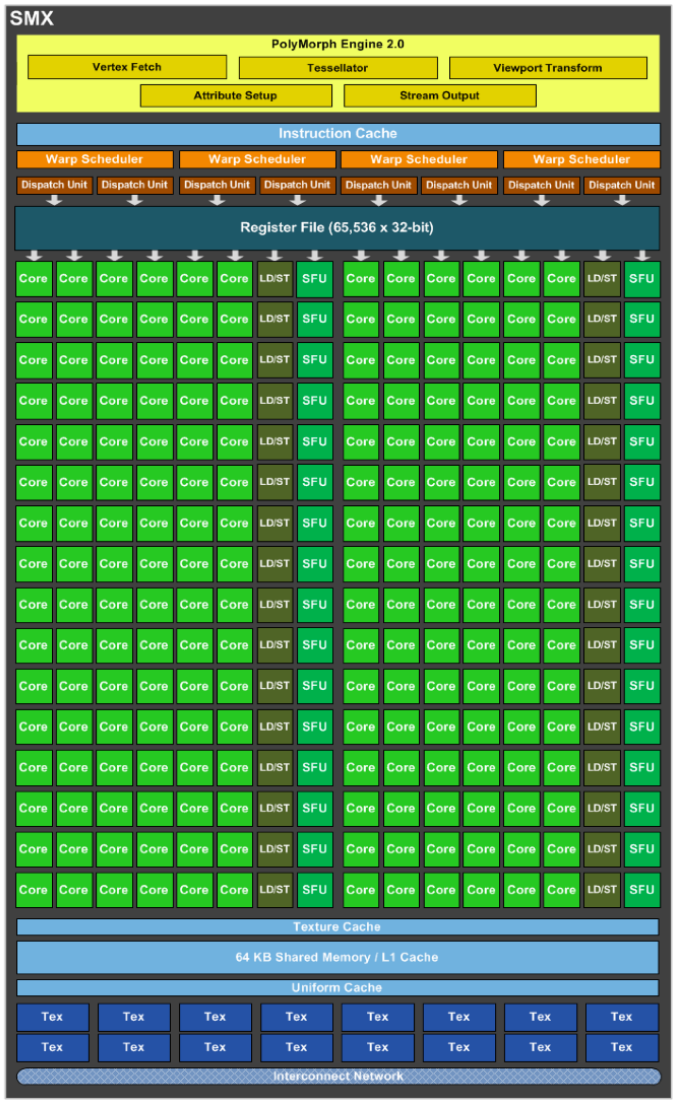
Kepler GeForce の構造

Nvidia white paper より引用



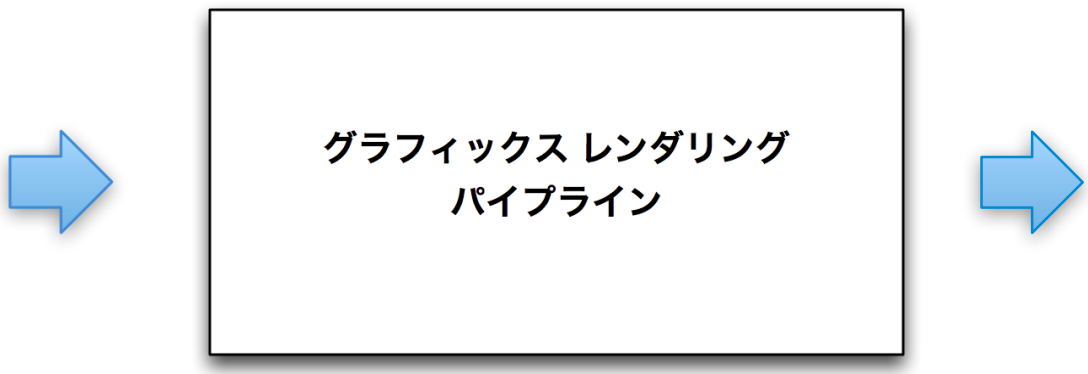
Kepler GeForce の構造

Nvidia white paper より引用



グラフィックス レンダリング パイプライン

Graphics rendering pipeline 「パイプライン」

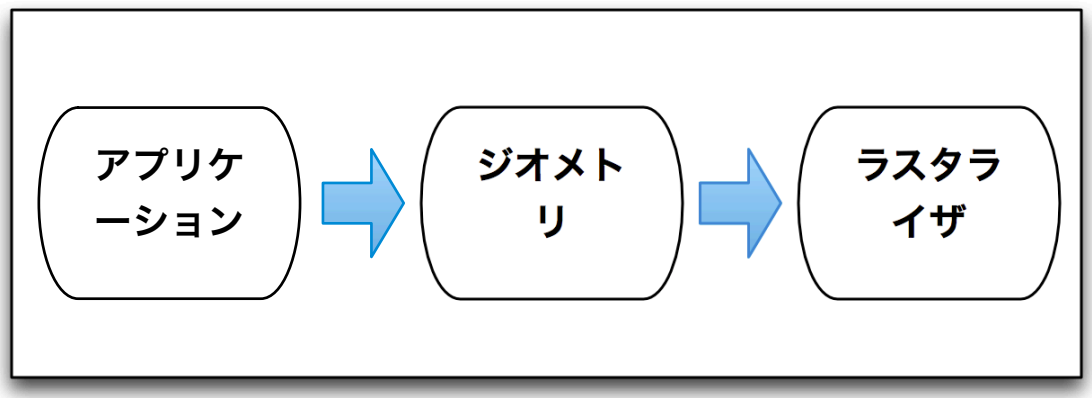


input: カメラ設定、3 次元オブジェクト、光源、ライティングモデル、テクスチャ

output: 2 次元画像

パイプラインの構造

アプリケーション ステージ → ジオメトリ ステージ→ ラスタライザ ステージ



アプリケーション ステージ

ソフトウェア

オブジェクトの生成、アニメーション、衝突検出、カリング、等々

ジオメトリ ステージ

ハードウェア

ビューボリューム

canonical view volume （標準ビューボリューム）

辺の長さ 2 の立方体

$$(-1,-1,-1) \leq (x,y,z) \leq (1,1,1)$$

標準ビューボリューム内の座標を正規化デバイス座標 （Normalized Device Coordinates, NDC）と呼ぶ。

ジオメトリステージの役割は、アプリケーションが設定した CG 世界をビューボリュームにマップすること。

射影

2 種類：

・正射影 orthographic projection, 平行投影（parallel projection）

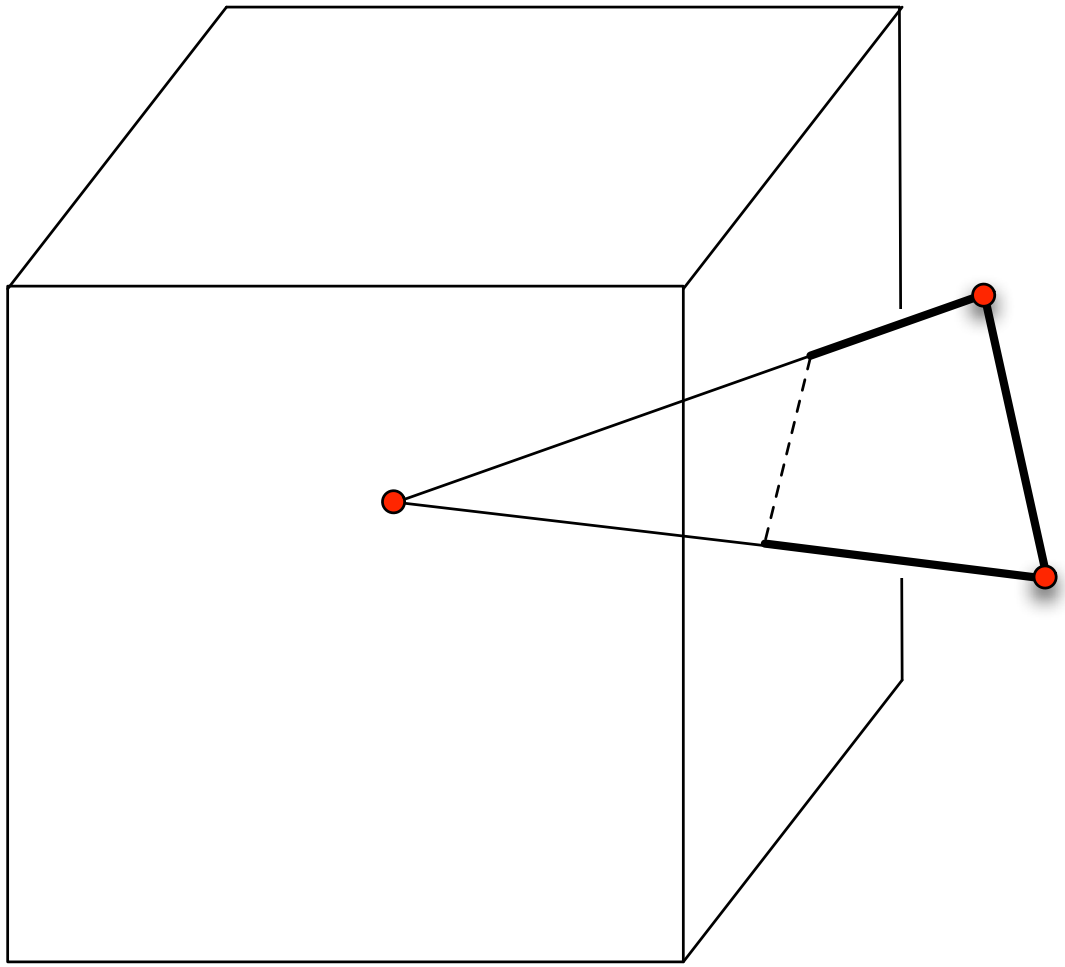
・透視射影 perspective projection, 遠近投影

視錐台 view frustum

どちらの射影も 4 行 4 列の行列演算で書ける（後述）。

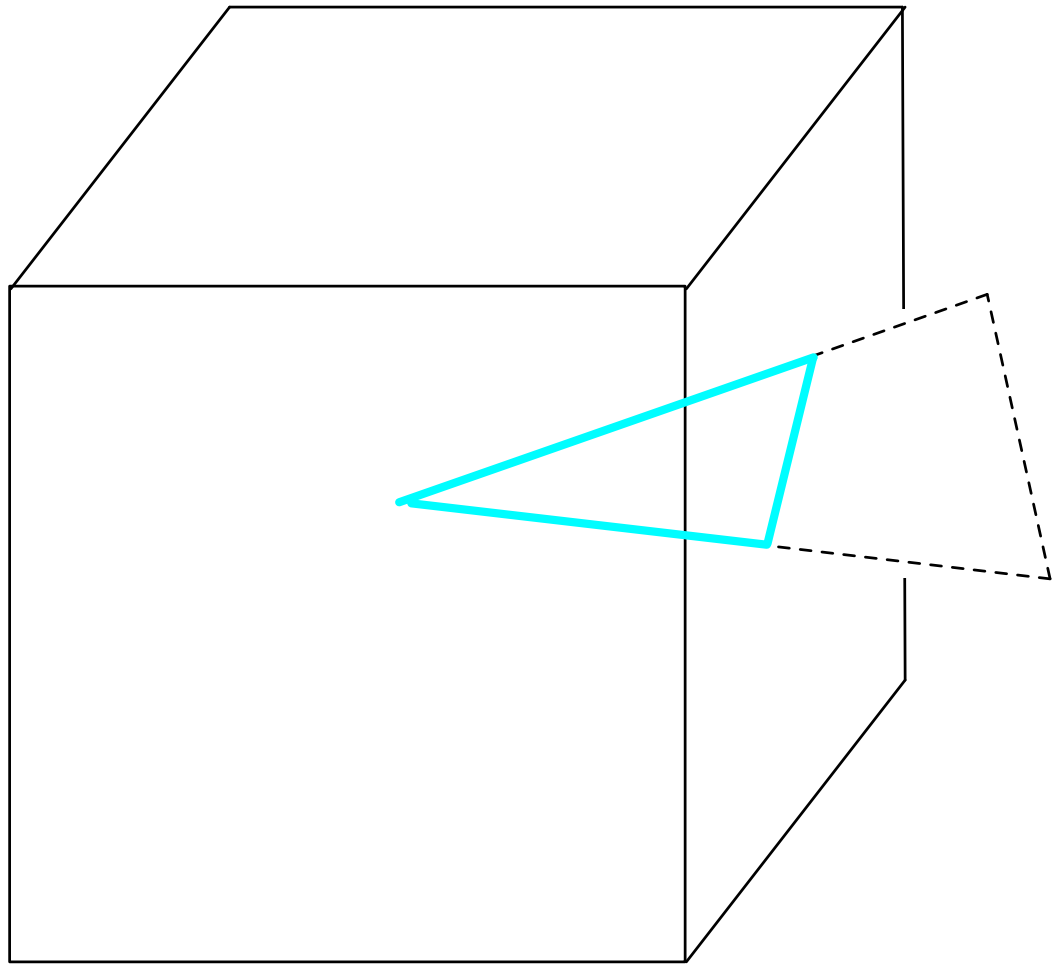
クリップ処理

ビューボリュームの外部は描かない。



クリップ処理

ビューボリュームの表面に新しい辺が自動的にできる。

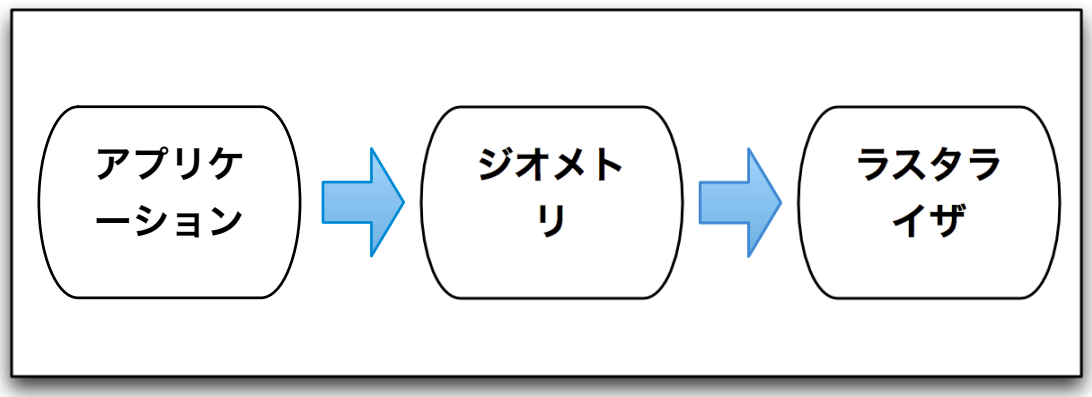


スクリーンマッピング

最終的な画像の大きさ（width と height）に合わせて、立方体のビューボリュームを x 方向と y 方向にスケール変換＋平行移動させる。z 方向には何もしない（$-1 \leq z \leq 1$）

パイプラインの構造

再掲

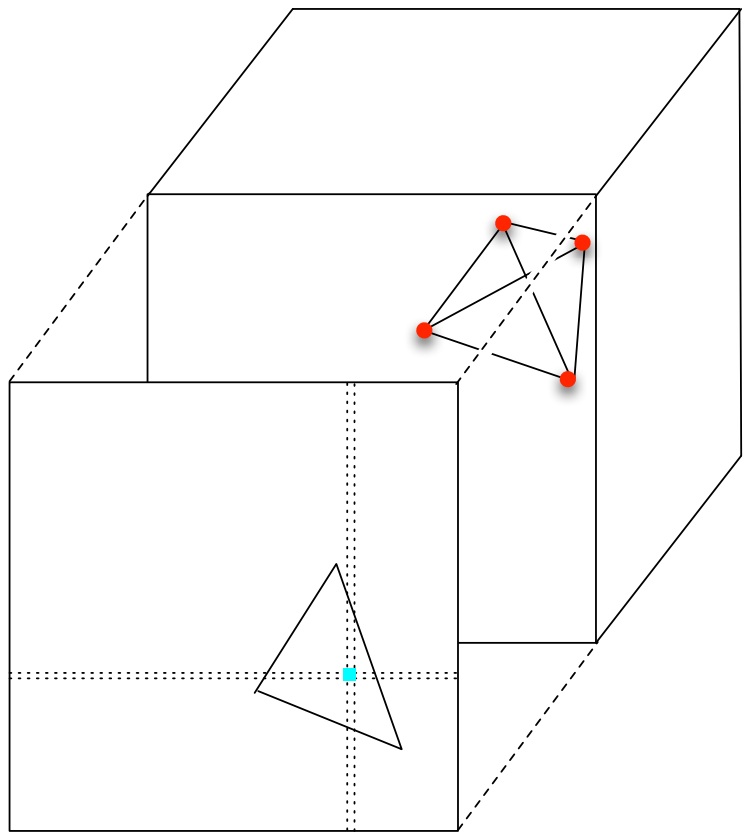


ラスタライザステージ

ビューボリューム内のデータ（3D）＋テクスチャデータ（2D） → 画像（2D）

頂点のデータ → ピクセルの色

簡単のため、スクリーンマッピングを省略すると



ラスタライザステージ

デプステスト。後述（p. 51）

アルファテスト

ステンシルテスト

テクスチャマッピング

グラフィックスハードウェア

グラフィックスハードウェアの概念図

→ 【図】

フレームバッファ

カラーバッファ

Z バッファ

ステンシルバッファ

カラーバッファ

ピクセル RGBA

16 bits, 23 bits, 32 bits

RGBA32 = R8 + G8 + B8 + A8

Z バッファ

Z-buffer

デプステスト

Edwin Catmull が考案。→ 写真

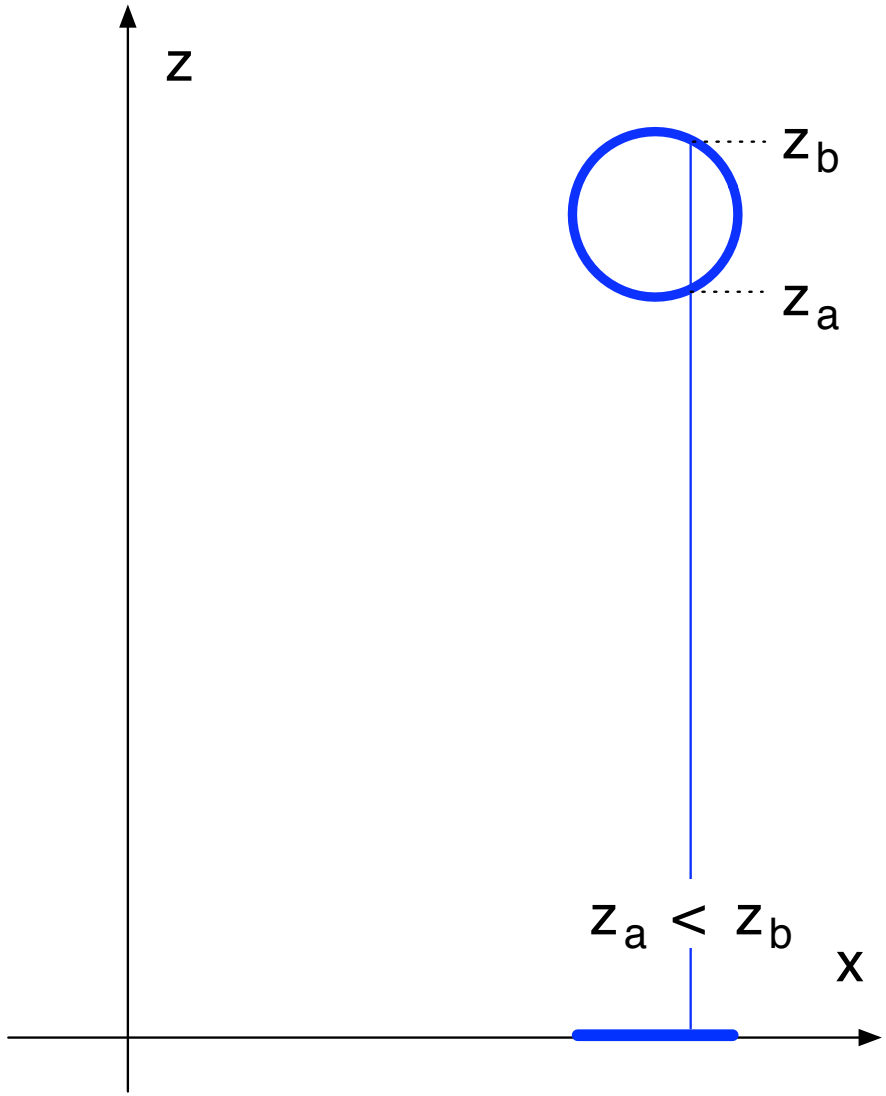
博士論文（1974 年）。

RenderMan 開発（アカデミー賞）

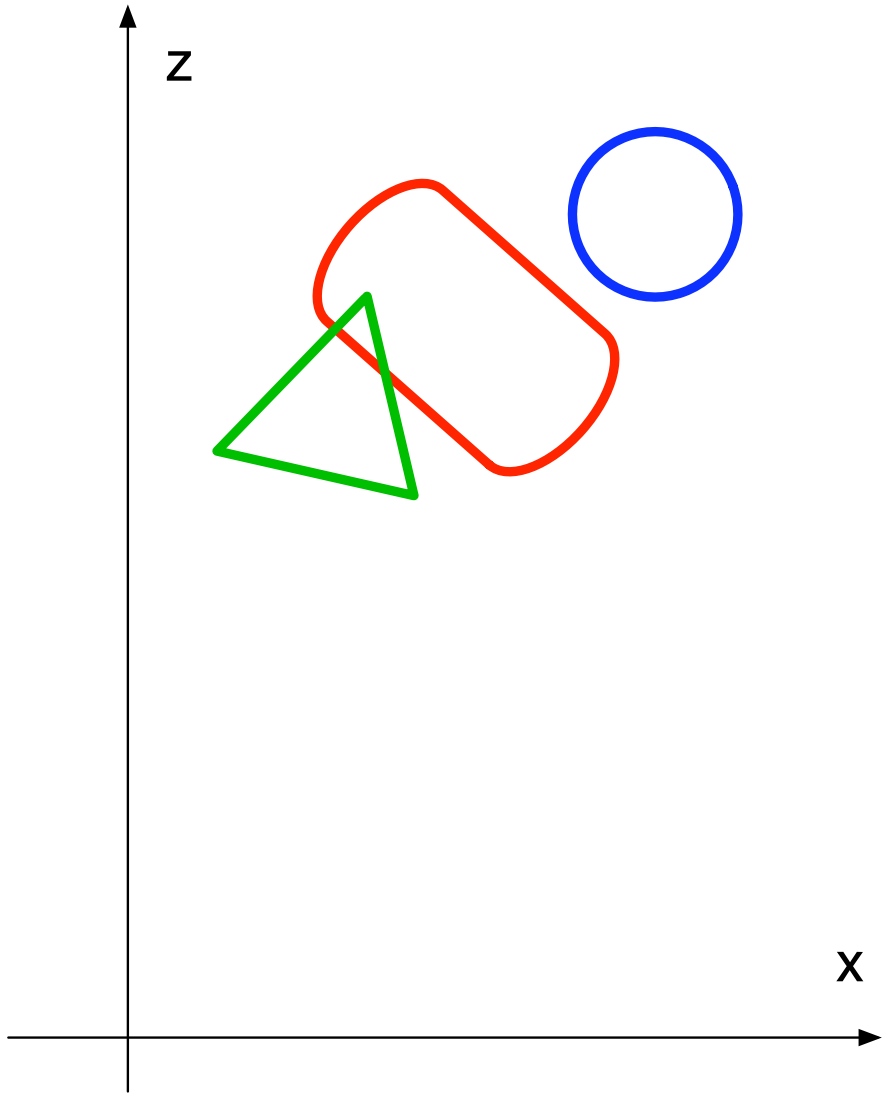
ルーカスフィルム → ピクサー設立。

ウォルト・ディズニー・アニメーション・スタジオ社長

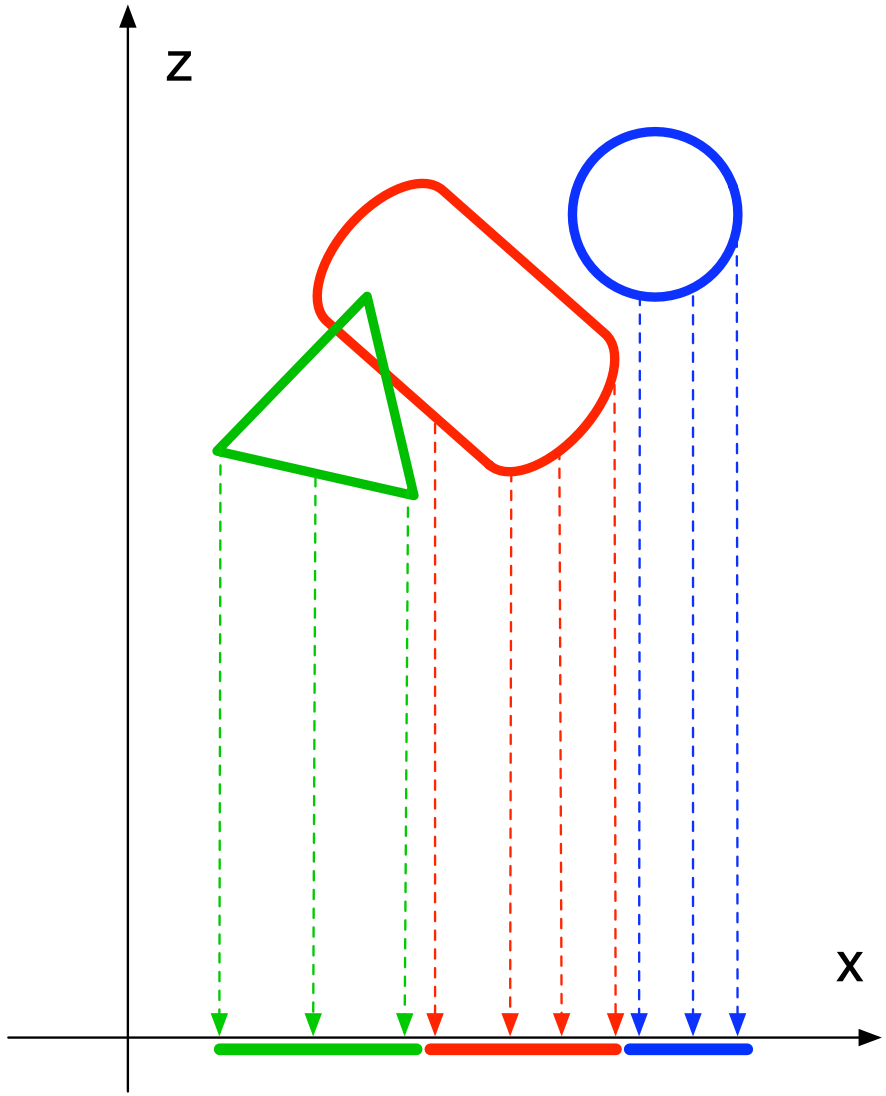
Z-buffer



Z-buffer

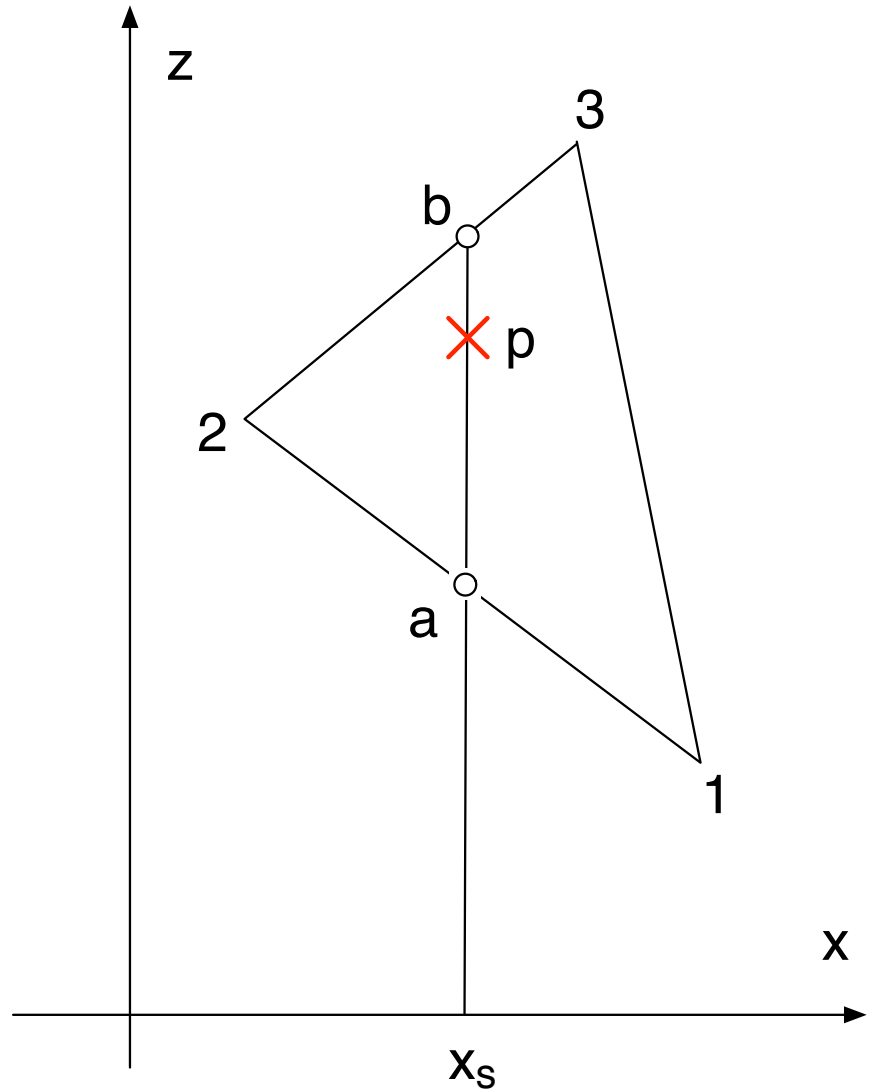


Z-buffer



Z-buffer

$$
\begin{aligned}
z_a &= z_1 + (z_2 - z_1)\frac{x_s - x_1}{x_2 - x_1} \\
z_b &= z_2 + (z_3 - z_2)\frac{x_s - x_2}{x_3 - x_2} \\
z_p &= z_a + (z_b - z_a)\frac{y_p - y_b}{y_b - x_a}
\end{aligned}
$$

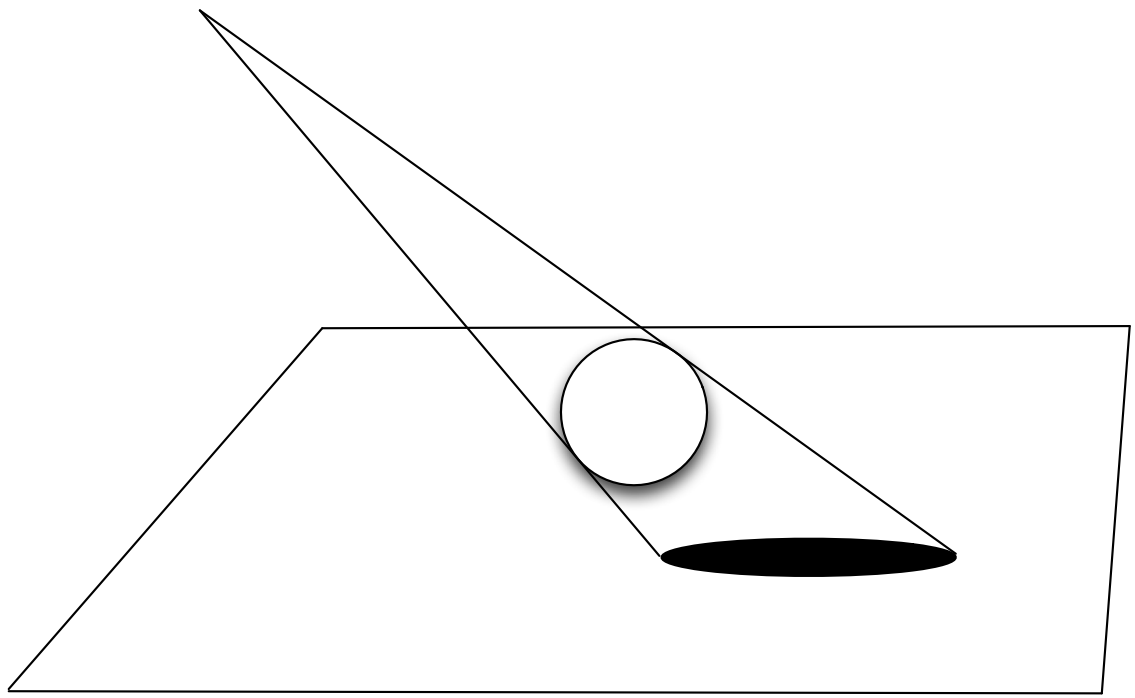


デプスマップアルゴリズム

シャドウマップアルゴリズムともいう (Williams1978)

光源に視点をおいてレンダリング

光源からオブジェクトの各点までの距離 → z バッファ → 2 次元データ → テクスチャ→ テクスチャマッピング



ステンシルバッファ

カラーバッファの値を実際に描画するかどうかをピクセル単位で制御

「型紙」