最新 OpenGL ハードウェアでのシャドウ マッピング

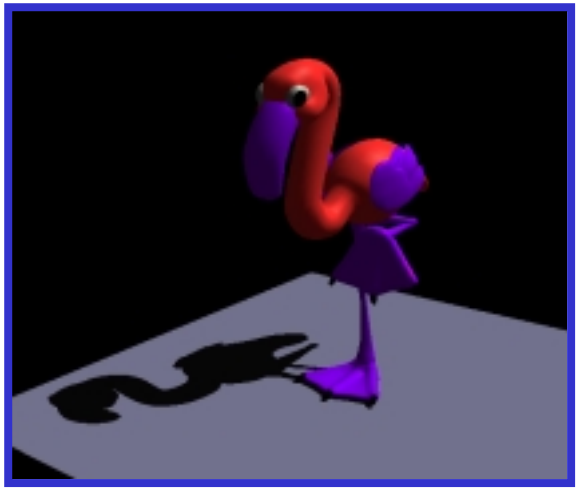
CEDEC 2001
Tokyo, Japan

Mark J. Kilgard
Graphics Software Engineer
NVIDIA Corporation

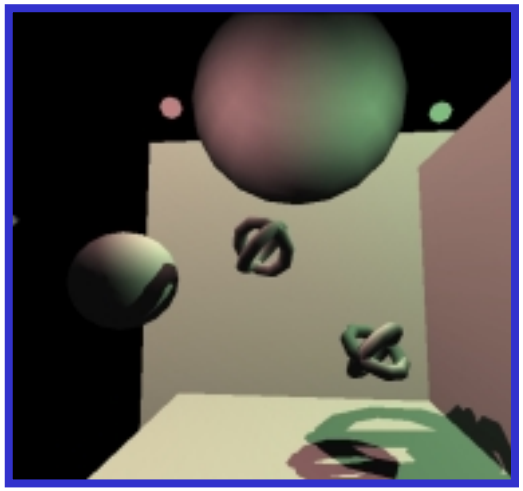
より高度なシャドウが求められる理由

- シャドウはシーンのリアリズムを高める
  - 現実の世界にはシャドウ (影) がある
  - ゲームの体感をさらにコントロール
    - ドラマチックさを出す効果
    - 不気味な感じを出す効果
  - 他の形態の芸術ではシャドウの価値が認められている
  - しかし、ほとんどのゲームにはまだリアルなシャドウは使われていない

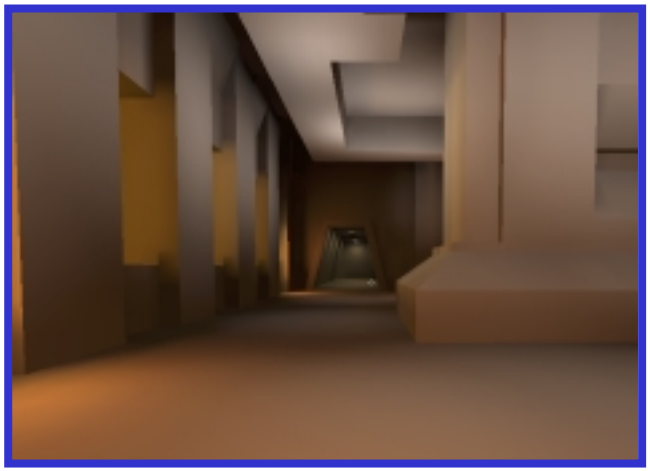
一般的なリアルタイム シャドウ テクニック



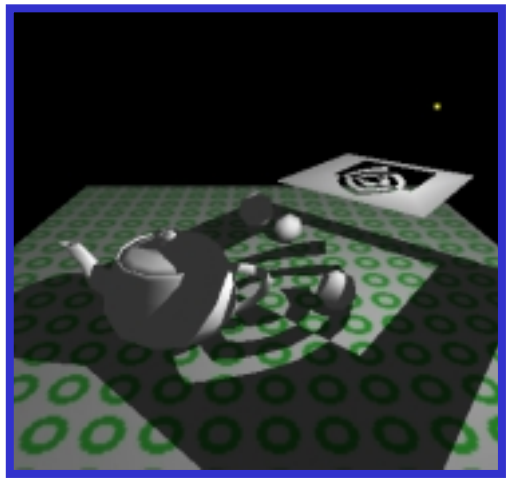
射影平面シャドウ



シャドウボリューム



ライト マップ



ハイブリッド アプローチ

NVIDIA

一般的なシャドウ テクニックの問題

- 実は制約が多い
  - 射影平面シャドウ
    - フラットなサーフェスに対してのみうまく機能
  - ステンシル シャドウ ボリューム
    - シャドウ ボリュームを決定するのは困難な作業
  - ライト マップ
    - 動的シャドウにはまったく不適
  - 総じて、対象を選ばない万能なシャドウ処理を見つけるのは難しい

もうひとつのテクニックを紹介：シャドウ マッピング

- イメージ空間のシャドウ決定
  - **Lance Williams** が **1978** 年に基本的なアイディアを発表
    - 偶然にも同じ年に **Jim Blinn** がバンプ マッピングを発明 **(** グラフィックスに関しては最上の年**)**
  - 完全にイメージ空間内のアルゴリズムとは
    - シーンのジオメトリに関する情報が不要ということ
    - エイリアシング対策が必要
  - よく知られているソフトウェア レンダリング テクニック
    - **Pixar** の **RenderMan** はこのアルゴリズムを使用
    - **Toy Story** などで使われた基本的なシャドウイング テクニック

シャドウ マッピングの参考文献

- 重要な **SIGGRAPH** 論文
  - **Lance Williams, “Casting Curved Shadows on Curved Surfaces,” SIGGRAPH 78**
  - **William Reeves, David Salesin, and Robert Cook (Pixar), “Rendering antialiased shadows with depth maps,” SIGGRAPH 87**
  - **Mark Segal, et. al. (SGI), “Fast Shadows and Lighting Effects Using Texture Mapping,” SIGGRAPH 92**

シャドウ マッピングの概念 (1)

- ライトの視点からの深度テスト
  - 2 パス アルゴリズム
  - まず、光源からのビューで深度バッファをレンダリング
    - 結果は “深度マップ” または “シャドウ マップ”
    - 基本的には、ライトに最も近いピクセルの深度を示す 2D 関数
  - この深度マップを 2 番目のパスで使用

シャドウ マッピングの概念 (2)

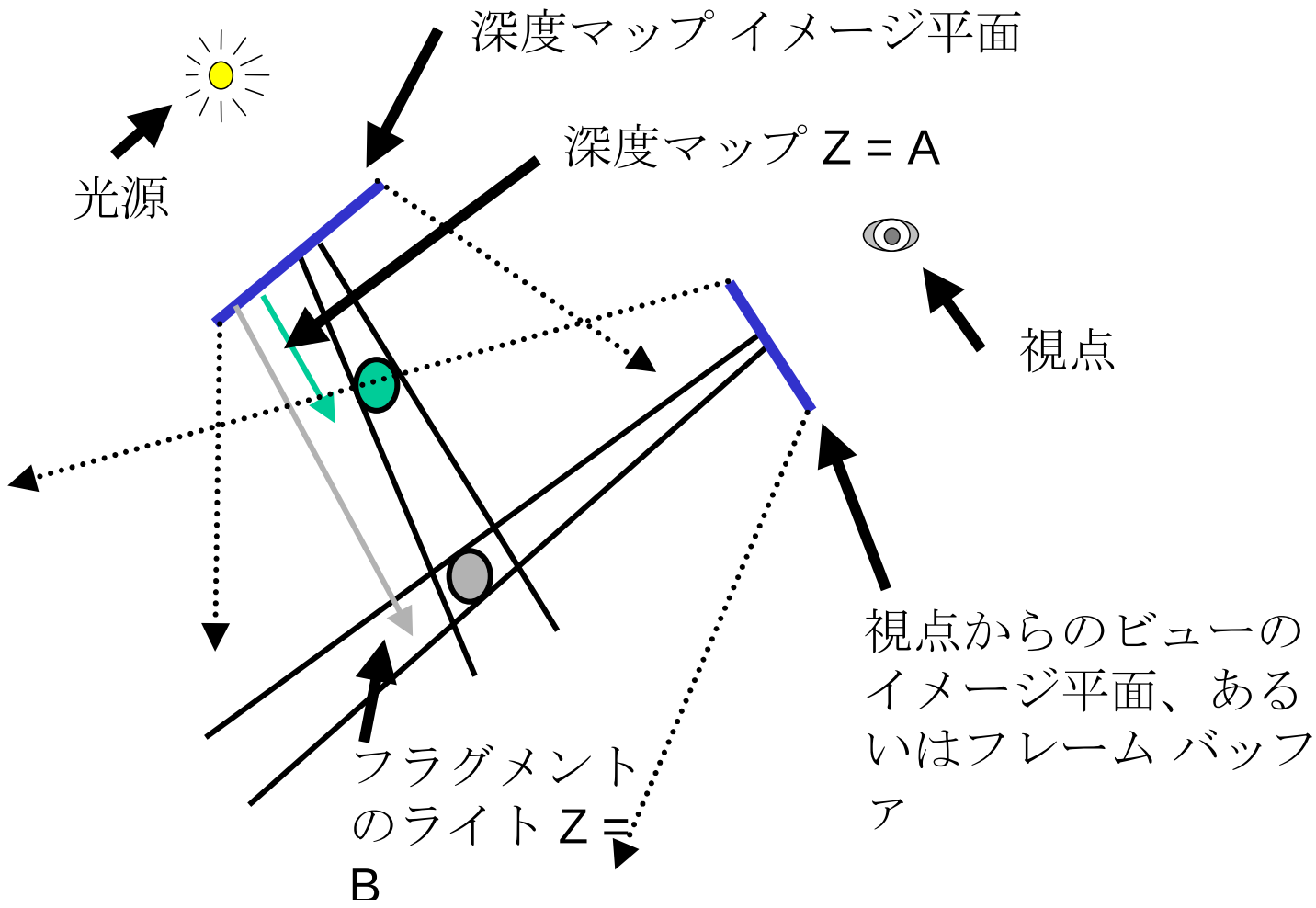
- 深度マップを使用したシャドウの決定
  - 次に、視点からのビューでシーンをレンダリング
  - ラスタ化した各フラグメントについて：
    - ライトに対して相対的なフラグメントの **XYZ** 位置を決定
    - このライトの位置は、深度マップの作成に使われた錐台と一致するように設定する必要がある
    - 深度マップでライトの位置 **XY** にある深度の値とフラグメントのライトの位置 **Z** を比較

シャドウ マッピングの概念 (3)

- シャドウ マップ比較
  - 2 つの値
    - A = フラグメントのライトの位置 XY における深度マップの Z 値
    - B = フラグメントのライトの位置 XYZ の Z 値
  - B が A より大きい場合は、フラグメントよりも光源に近いところに何かがある
    - この場合、フラグメントがシャドウになる
  - A と B がほぼ等しい場合、フラグメントはシャドウにならない

2D のピクチャで示したシャドウ マッピング (1)

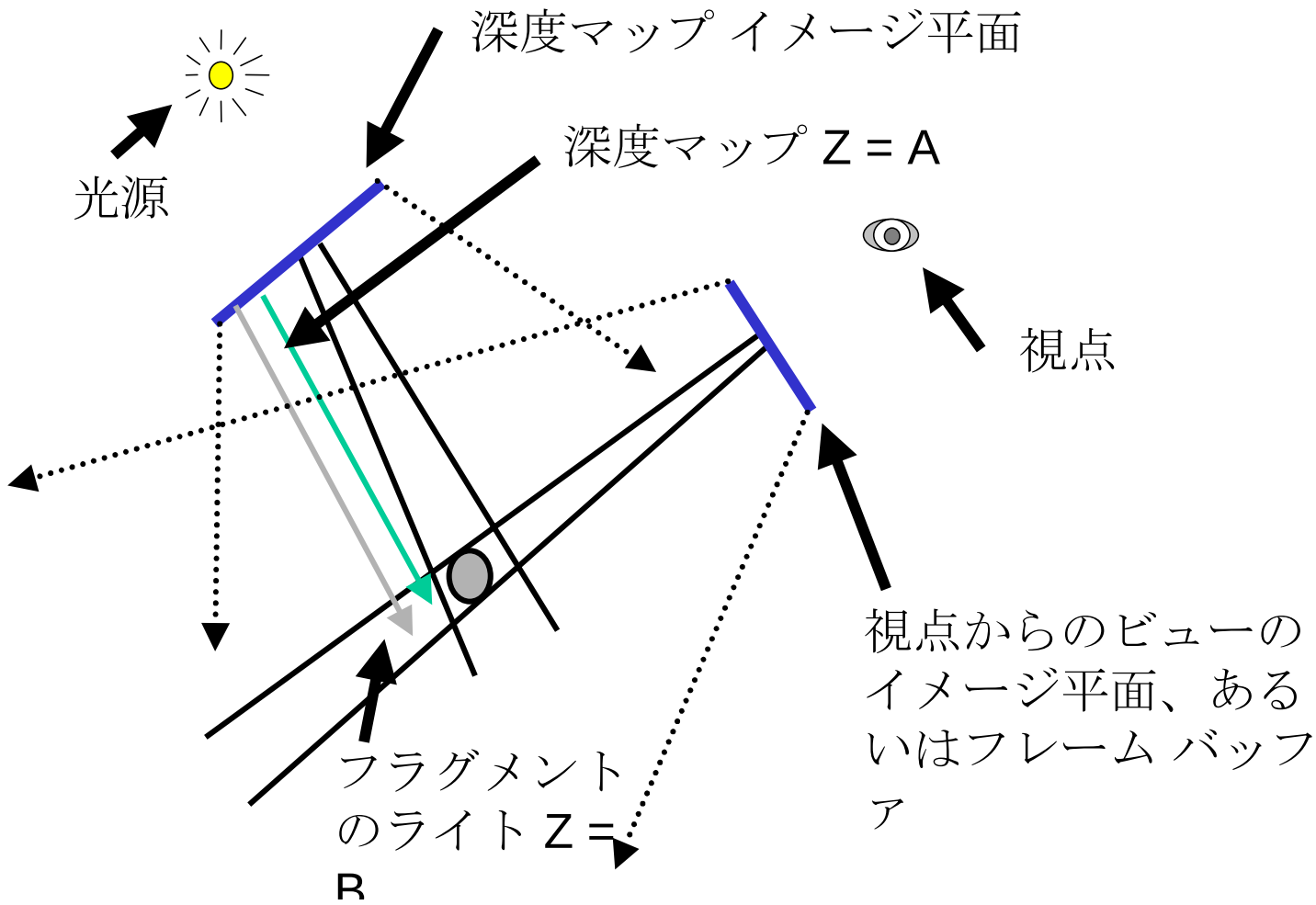
フラグメントがシャドウになる **A < B** の場合



NVIDIA

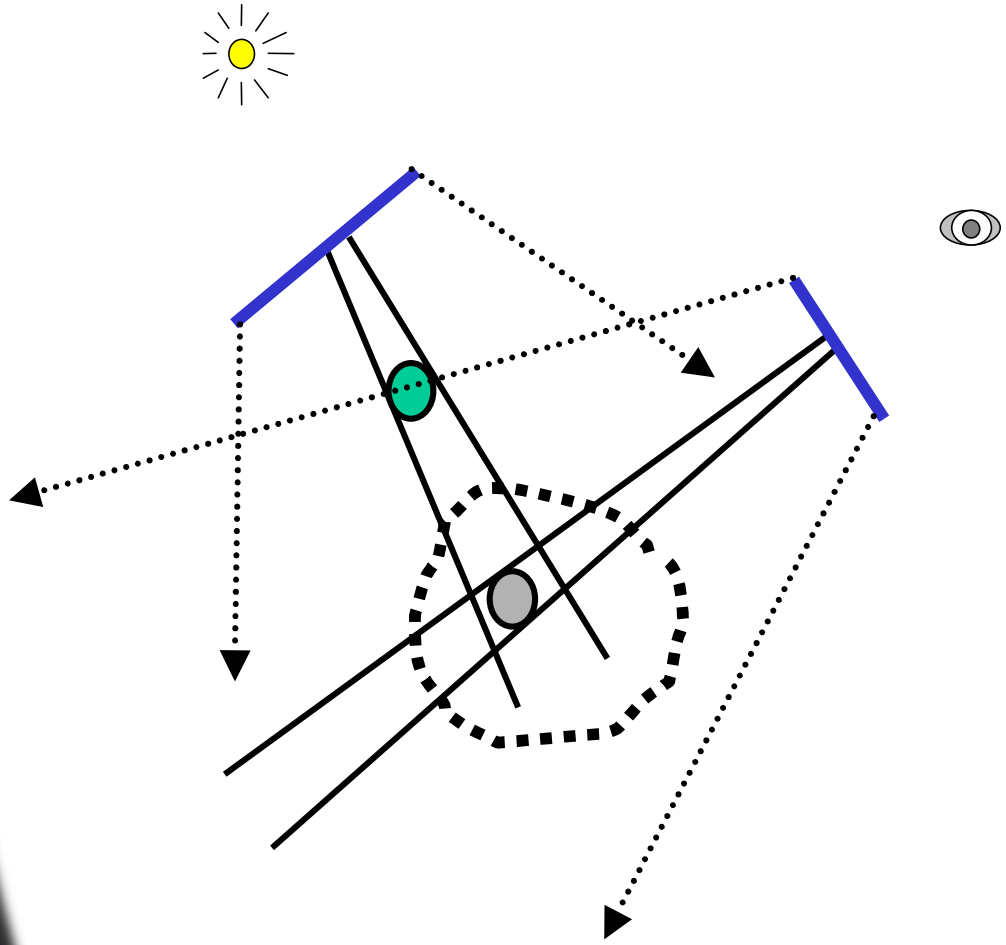
2D のピクチャで示したシャドウ マッピング (2)

フラグメントがシャドウにならない **A ≅ B** の場合



2D のピクチャで示したシャドウ マッピング (3)

イメージ精度の不一致に注意！



深度マップはフレーム バッファと解像度が異なる場合がある

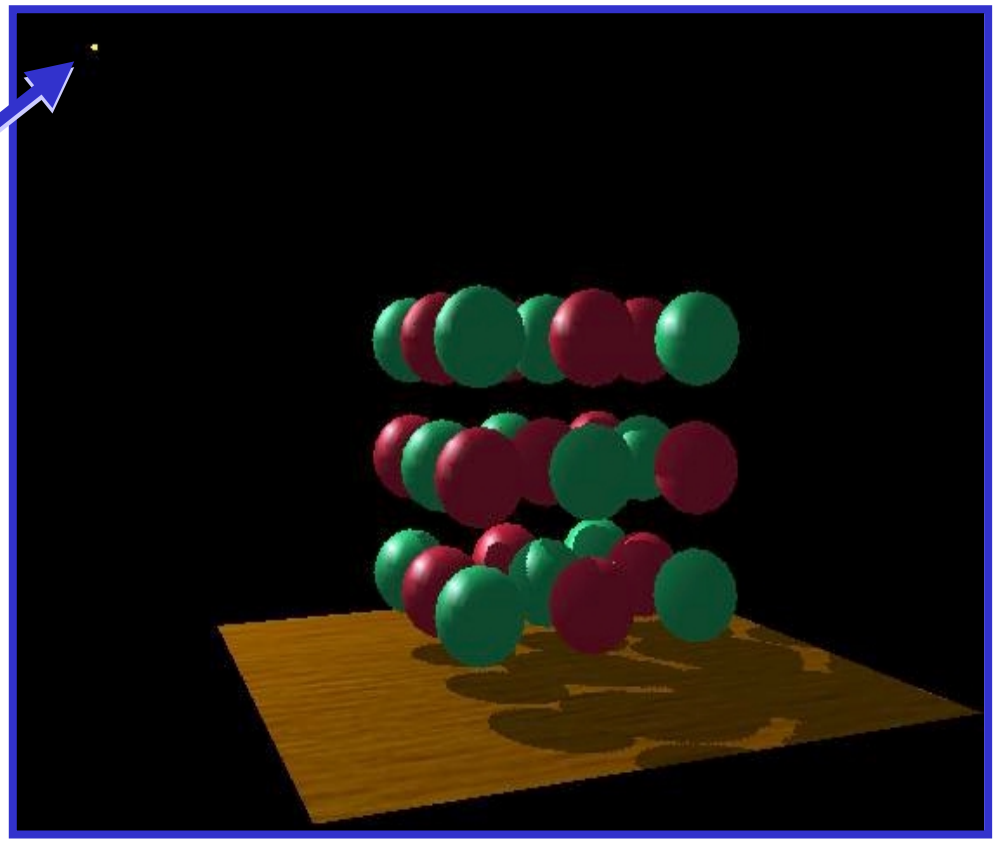
この不一致により不自然な効果が発生することがある

NVIDIA

シャドウ マッピング テクニックの可視化 (1)

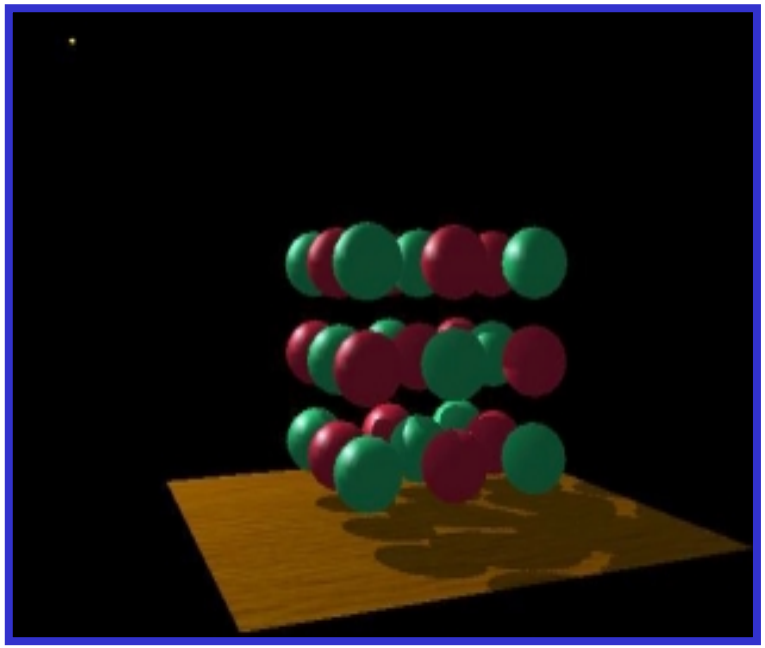
- シャドウを使ったかなり複雑なシーン

点光源

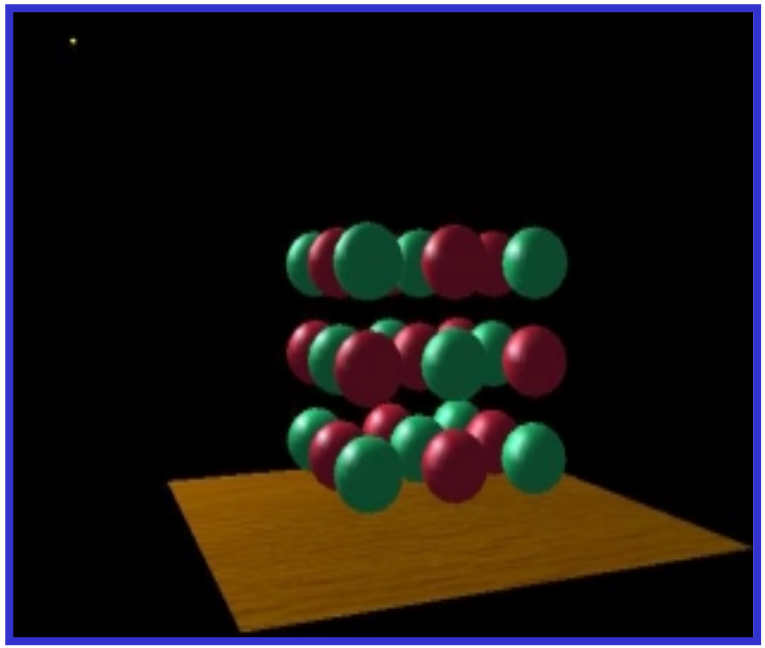


シャドウ マッピング テクニックの可視化 (2)

- シャドウを使ったシーンと使わないシーンの比較



シャドウあり

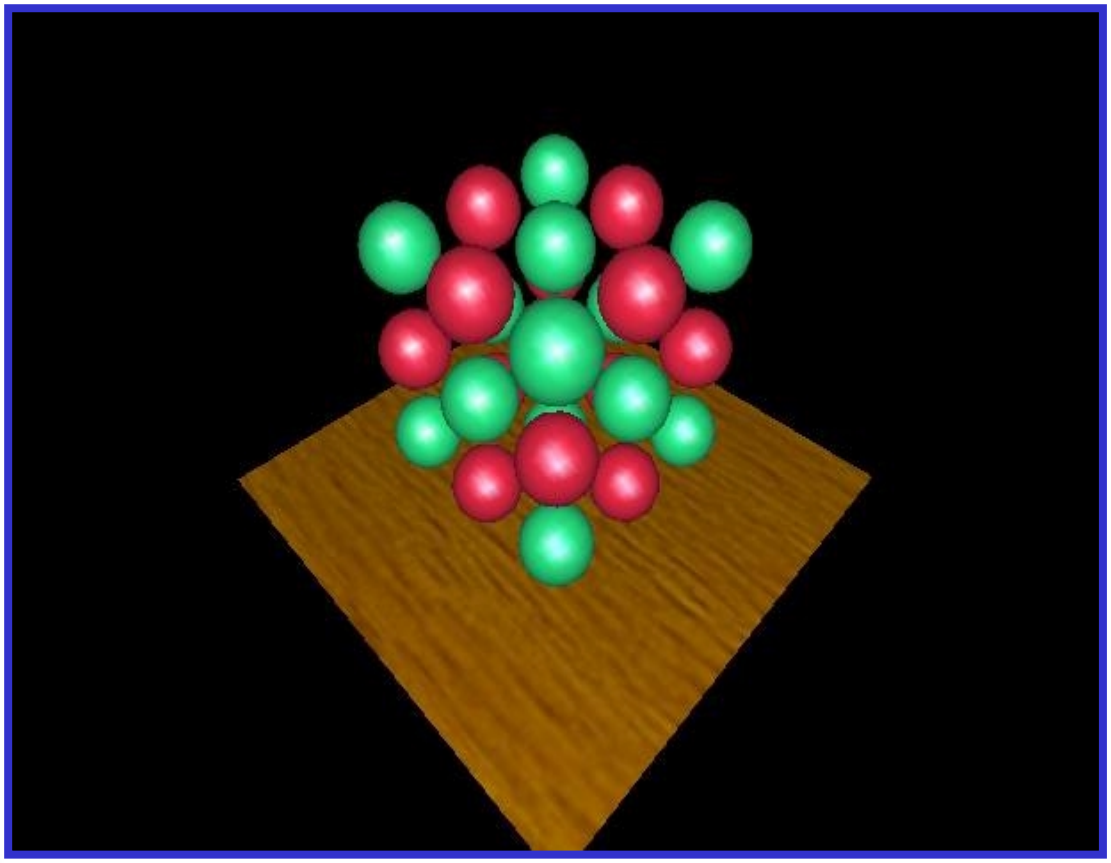


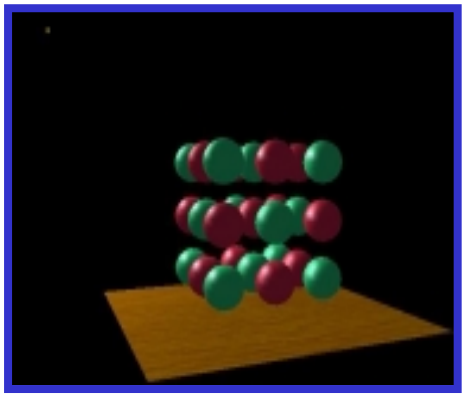
シャドウなし

NVIDIA.

シャドウ マッピング テクニックの可視化 (3)

- ライトのビューから見たシーン

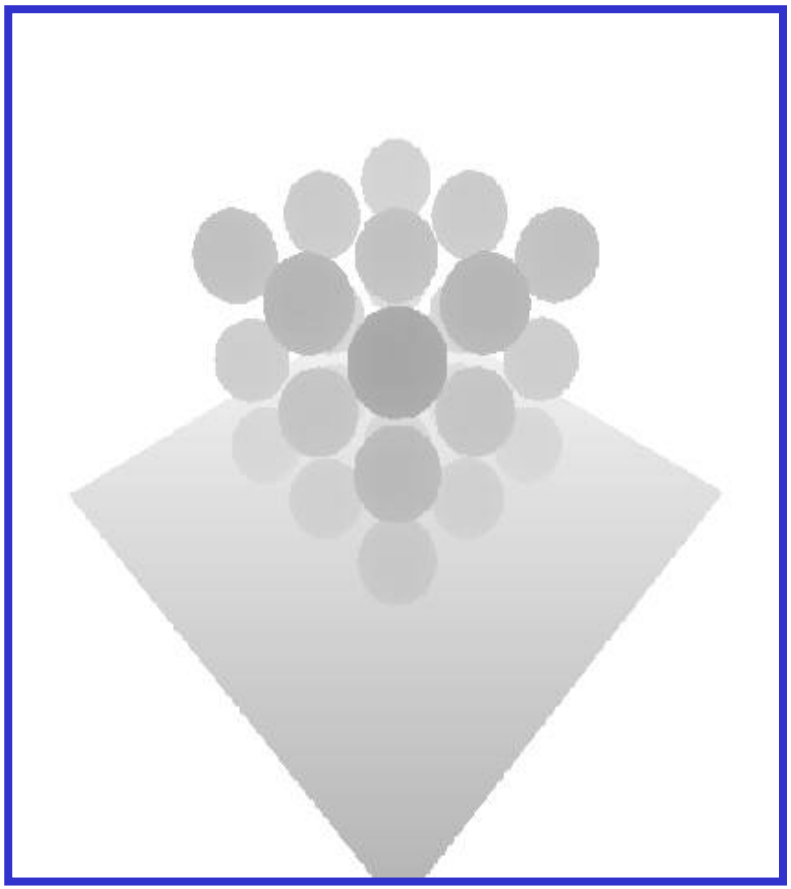


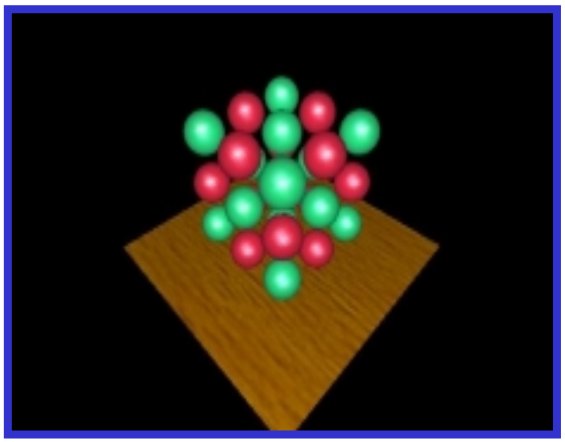


参考：目のビューから見たシーン

シャドウ マッピング テクニックの可視化 (4)

- ライトのビューから見た深度バッファ

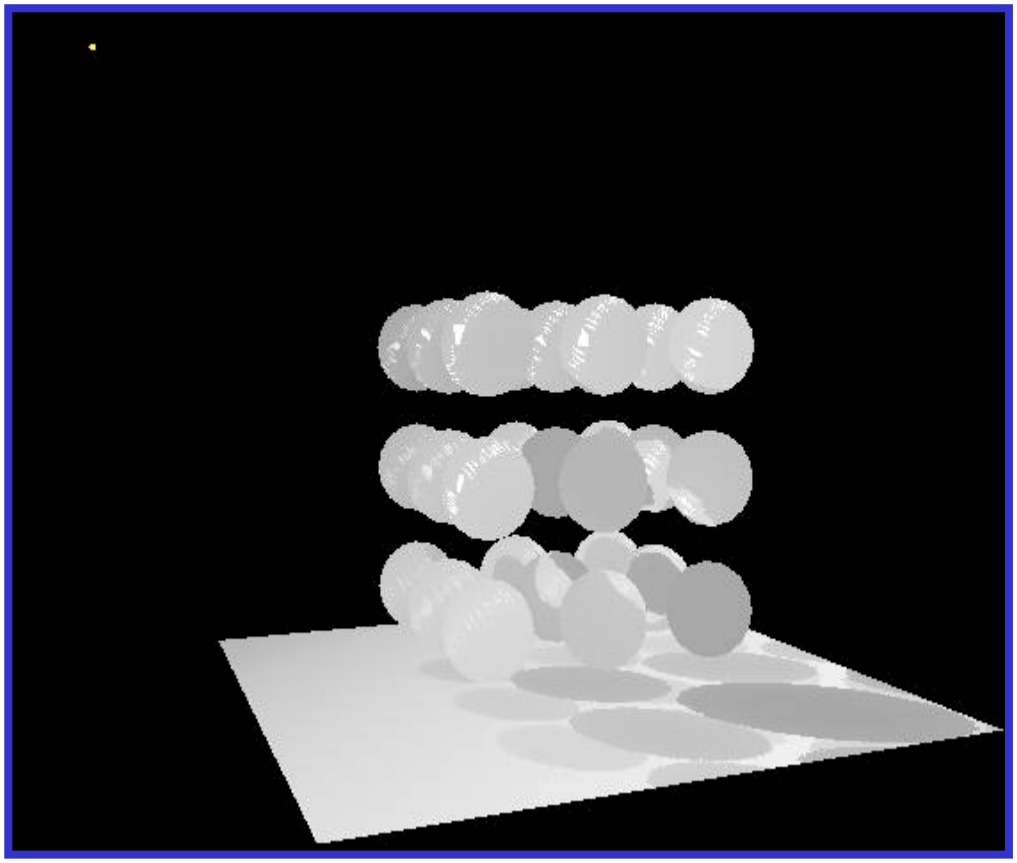


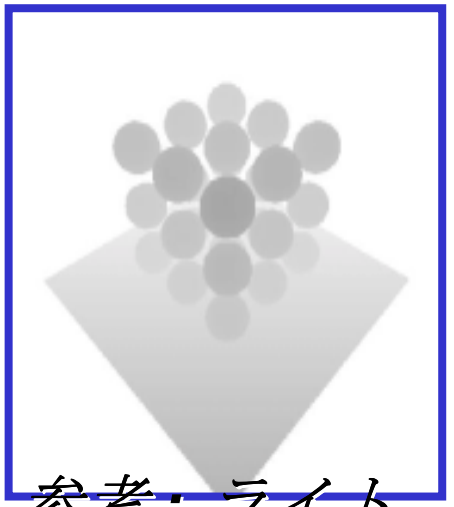


参考：ライトのビューから見たシーン

シャドウ マッピング テクニックの可視化 (5)

- 視点のビューに深度マップを射影

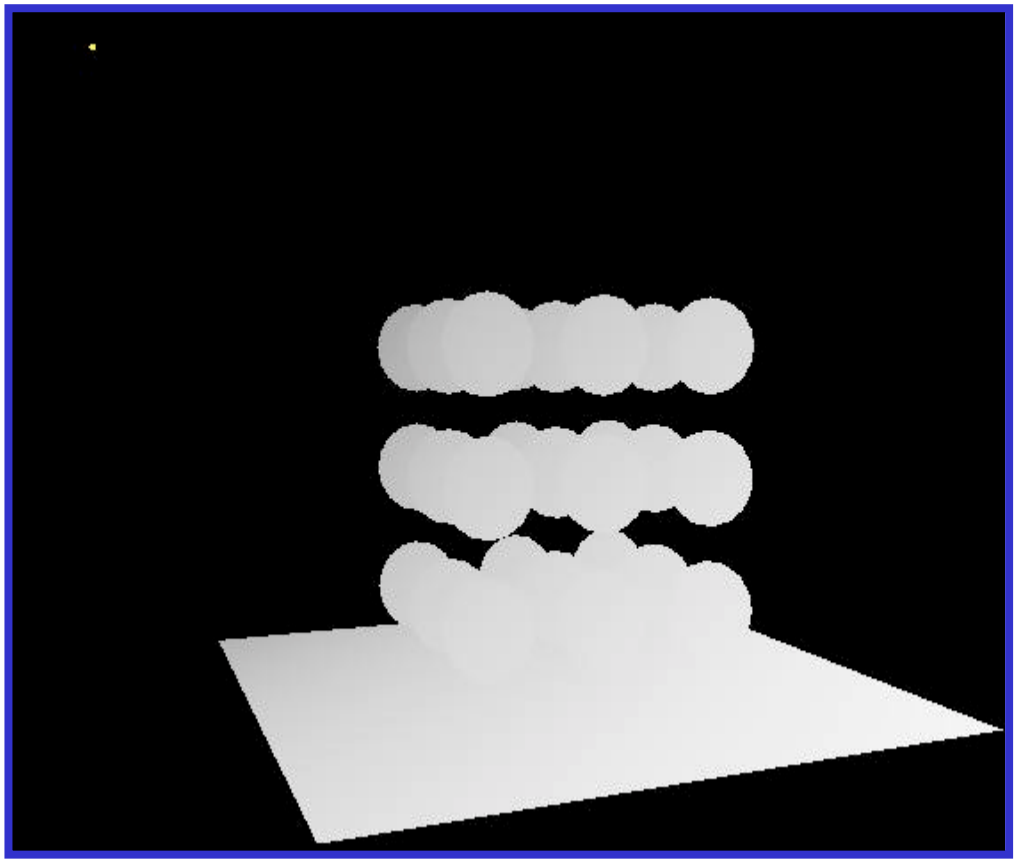




参考：ライトのビューから見た深度マップ

シャドウ マッピング テクニックの可視化 (6)

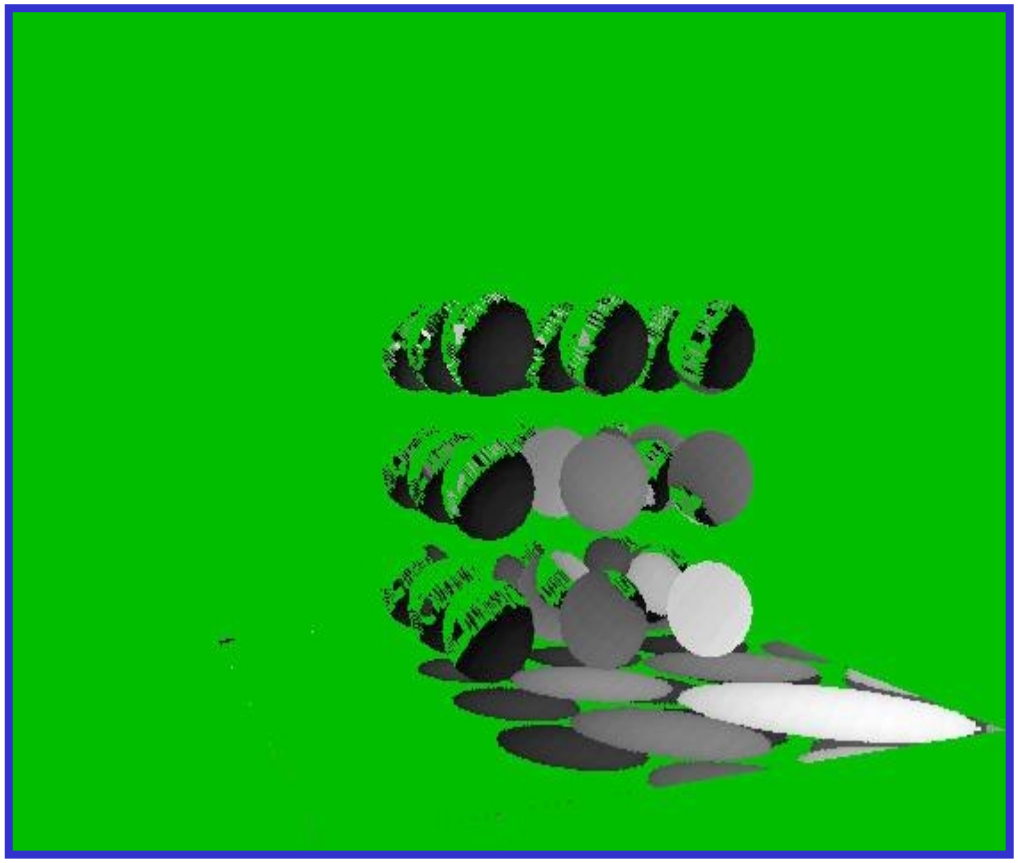
- ライトの平面距離を視点のビューに射影



シャドウ マッピング テクニックの可視化 (6)

- ライトの距離とライトの深度マップを比較

緑の部分は、ライトの平面距離とライトの深度マップがほぼ等しい



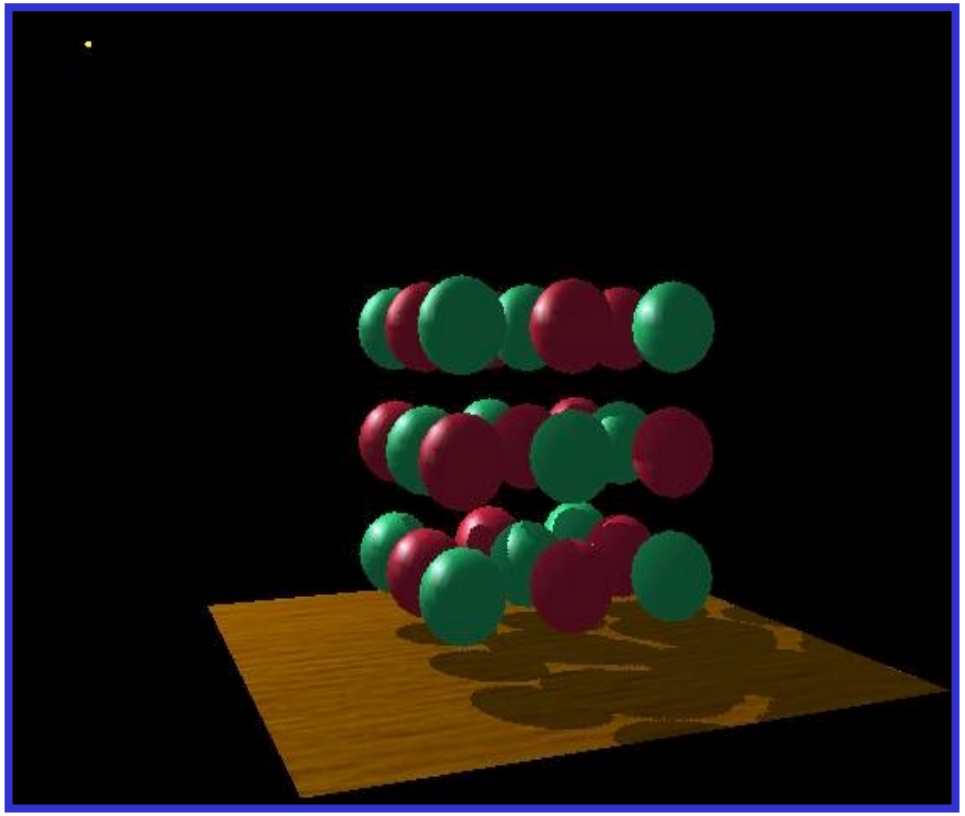
緑以外の部分が、シャドウになる部分

NVIDIA

シャドウ マッピング テクニックの可視化 (7)

- シャドウを使ったシーン

シャドウにスペキュラ ハイライトが現れることはない点に注意



曲面が互いにシャドウを落としている点に注意

NVIDIA

ライト ビュー深度マップの作成

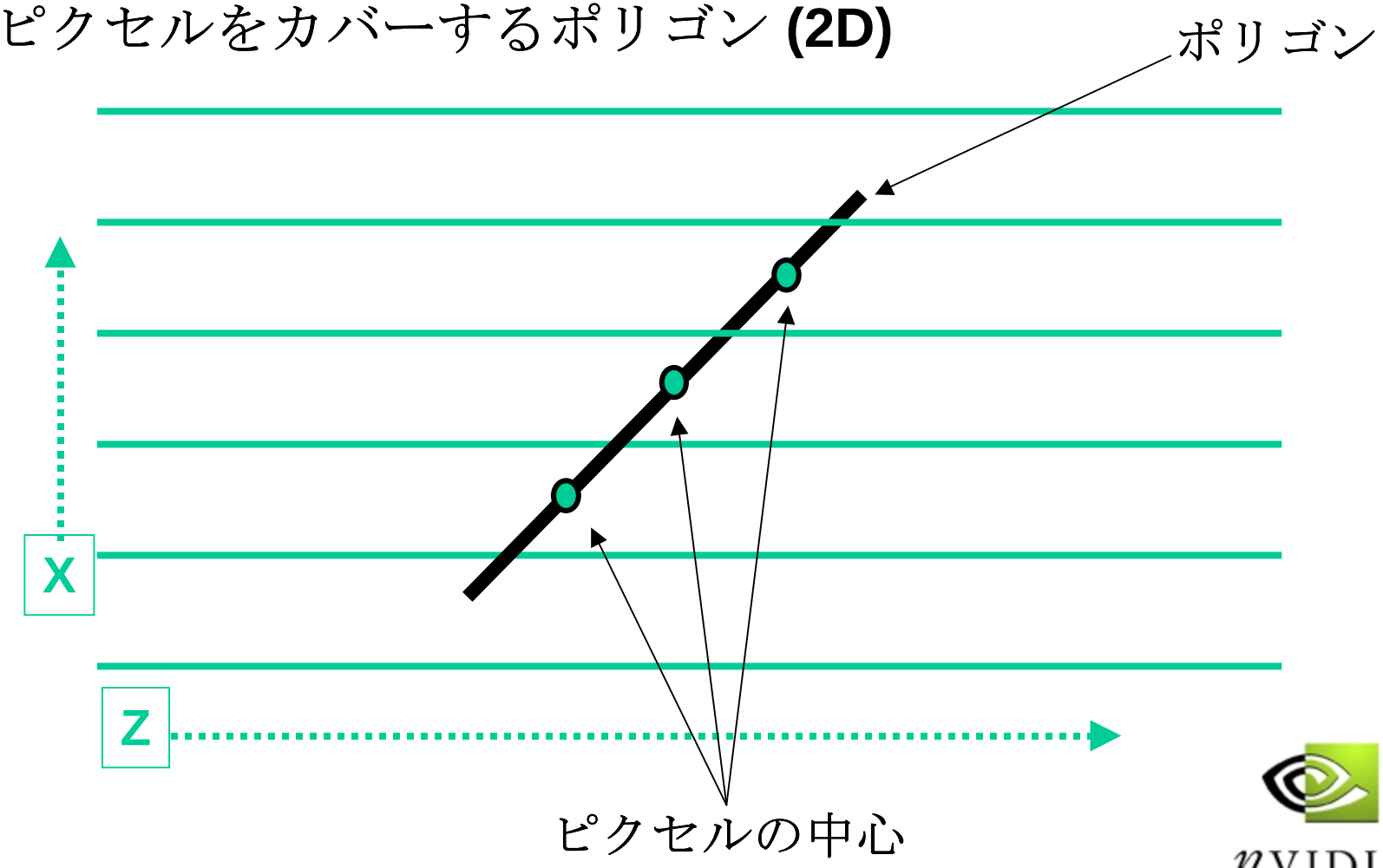
- 理論を実践で実現
  - 深度マップの作成
    - 既存のハードウェア深度バッファを使用
    - **glPolygonOffset** を使用して深度値をオフセットして戻す
    - 深度バッファの内容を読み戻す
  - 深度マップは **2D** テクスチャにコピー可能
    - 残念ながら、深度値には、テクスチャで一般的に使われる **8** ビットよりも高い精度が要求されることが多い
    - 深度の精度は通常 **16** ビットまたは **24** ビット

シャドウ マップ作成時に **glPolygonOffset** を使用する理由

- 深度バッファには“ウィンドウ空間”の深度値が格納
  - パースペクティブ後の除算とは、非線形分布を意味する
  - **glPolygonOffset** はウィンドウ空間オフセットであることが保証されている
- “クリップ空間” **glTranslatef** の実行では不十分
  - シャドウ マッピングの実装でよくある間違い
  - 深度バッファ ユニットの実際のバイアスは錐台によって変化
  - ポリゴンの傾斜を計算に入れる方法がない

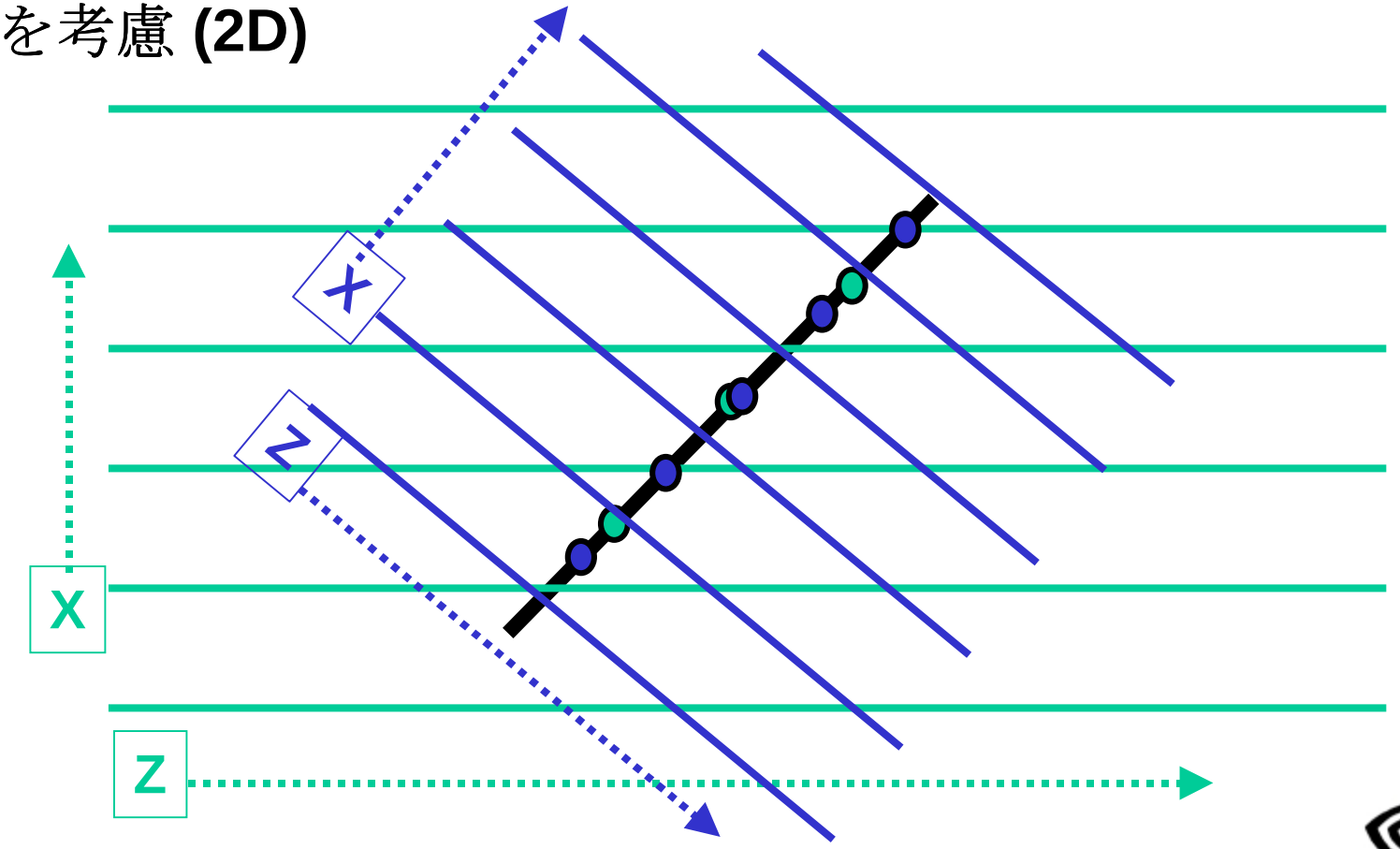
ピクセル中心でのポリゴンの深度のサンプリング **(1)**

- ピクセルをカバーするポリゴン **(2D)**



ピクセル中心でのポリゴンの深度のサンプリング (2)

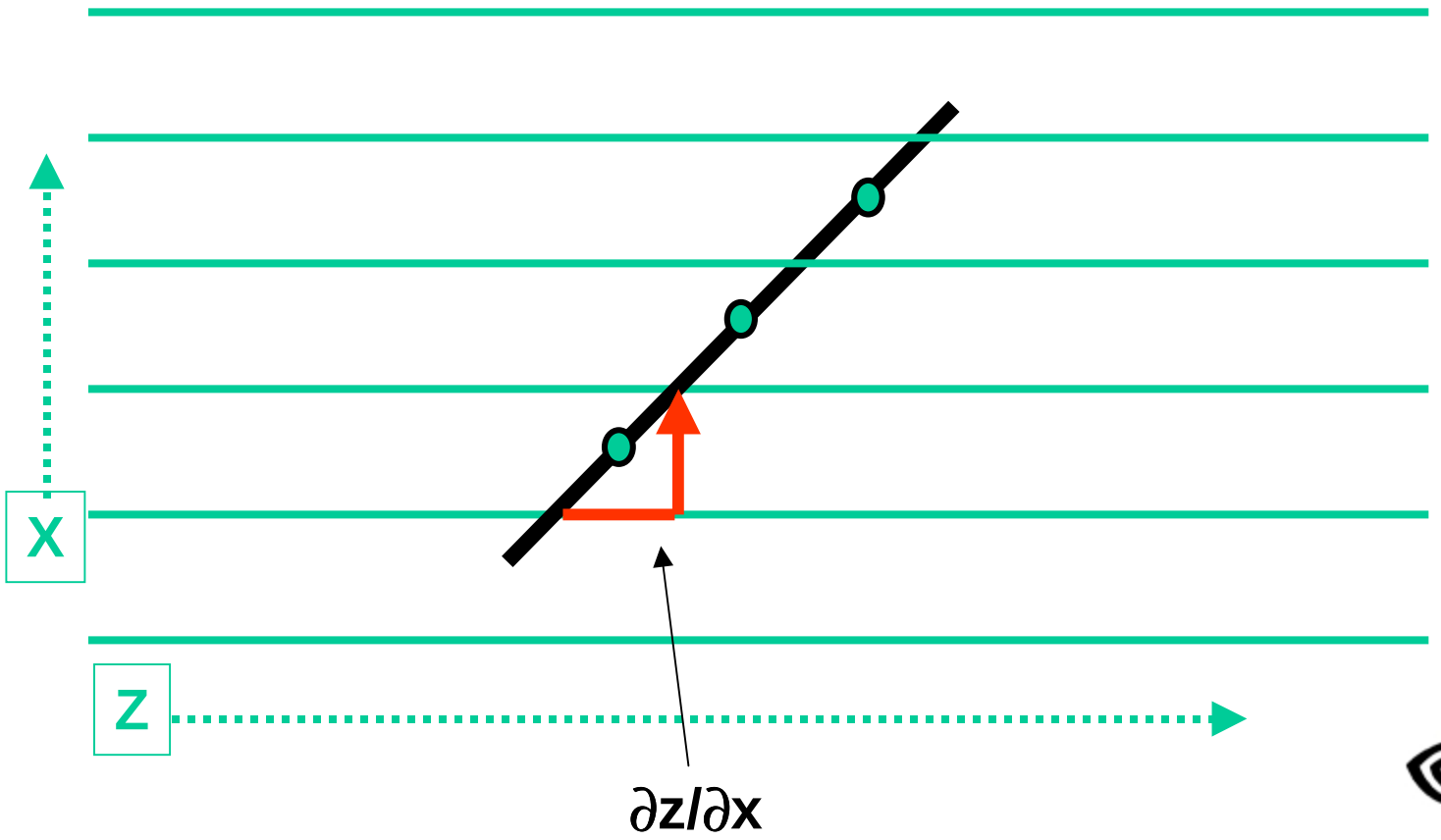
- ピクセルをカバーするポリゴンに対し第 **2** のグリッドを考慮 **(2D)**



NVIDIA

ピクセル中心でのポリゴンの深度のサンプリング (3)

- **X** に対して相対的な **Z** の変化

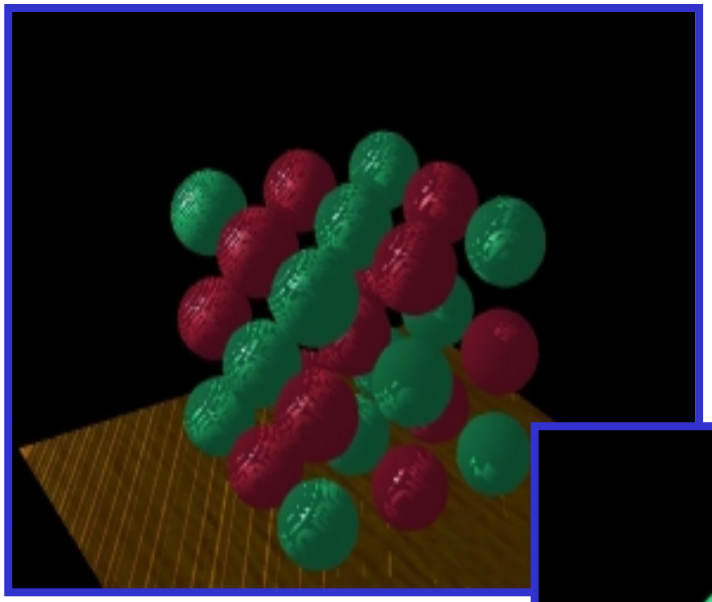


glPolygonOffset の傾斜が必要な理由

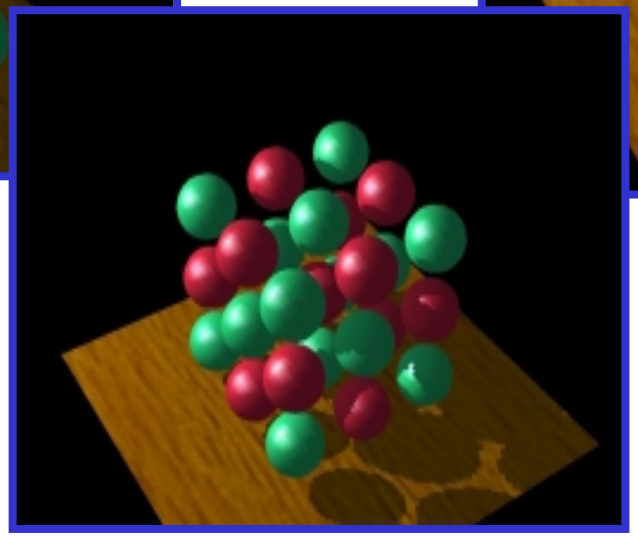
- ピクセルの中心が別のグリッドで再サンプリングされる場合
  - たとえばシャドウ マップ テクスチャのグリッドなど
- 再サンプリングされた深度は、たとえば
  +/-0.5 ∂z/∂x および **+/-0.5 ∂z/∂y** だけ離れている
- 最大絶対エラーは
  | 0.5 ∂z/∂x | + | 0.5 ∂z/∂y | ≈ max(| ∂z/∂x | , | ∂z/∂y |)
  - **2** つのグリッドのピクセル フットプリント エリア比が **1.0** であると想定
  - そうでない場合は、エリア比でのスケーリングが必要
- ポリゴン オフセットの **“傾斜”** 深度バイアスは厳密には何をするのか

深度マップ バイアスの問題

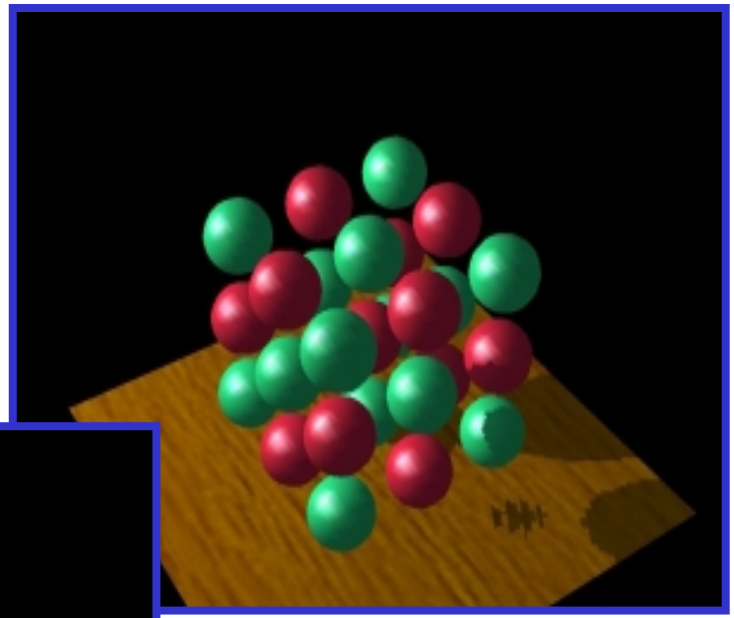
- ポリゴン オフセット バイアスの大きさは一定でない



バイアスが小さすぎると、シャドウが多すぎる



適度



バイアスが大きすぎると、シャドウが奥に遠くなりすぎる

深度マップ バイアスの選択

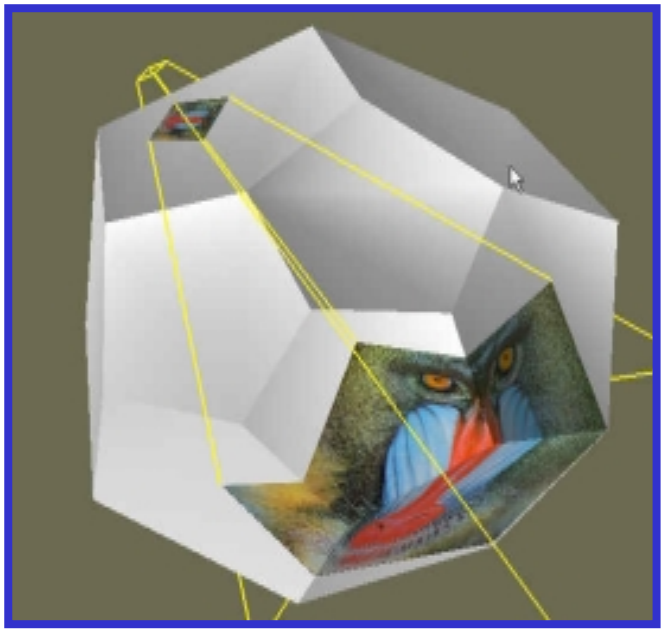
- それほど難しくはない
  - 通常は以下の通りでうまく機能する
    - **glPolygonOffset(scale = 1.1, bias = 4.0)**
  - 通常はバイアスが大きすぎるエラーの方がまし
    - シーンのシャドウの問題に応じて調整
  - シャドウ マップの精度にある程度依存
    - 精度が高いほど必要なバイアスは小さくなる
  - シャドウ マップを拡大する場合は、より大きいスケール値が必要なことが多い

シーンのレンダリングと深度テクスチャへのアクセス

- 理論を実践で実現
  - フラグメントのライトの位置は、視点線形テクスチャ座標生成によって生成できる
    - 具体的には **OpenGL** の **GL_EYE_LINEAR texgen**
    - ライト空間 **(x, y, z, w)** として同次 **(s, t, r, q)** テクスチャ座標を生成
    - **GeForce** などの **T&L** エンジンは **texgen** をアクセラレーション
    - 射影テクスチャを利用

射影テクスチャとは？

- 射影テクスチャの直感
  - スライドプロジェクタに類似



出典： ***Wolfgang Heidrich [99]***

NVIDIA

射影テクスチャについて (1)

- まず、パースペクティブ補正テクスチャとは何か？
  - 標準 **2D** テクスチャ マッピングでは **(s,t)** 座標を使用
  - **2D** パースペクティブ補正テクスチャ マッピング
    - **(s,t)** を視点空間で線形補間する必要があることを意味する
    - 頂点単位で **s/w**、**t/w**、**1/w** を計算
    - この **3** つのパラメータをポリゴン上で線形補間
    - フラグメント単位の計算 **s' = (s/w) / (1/w)** および **t' = (t/w) / (1/w)**
    - 結果はフラグメント単位のパースペクティブ補正 **(s', t')** となる

射影テクスチャについて (2)

- では、射影テクスチャとは？
  - 次に同次テクスチャ座標を考える
    - **(s, t, r, q) --> (s/q, t/q, r/q)**
    - **(x, y, z, w) = (x/w, y/w, z/w)** の同次クリップ座標に類似
  - 考え方としては、**(s/q, t/q, r/q)** をフラグメント単位で射影
  - そのためにはフラグメント単位の除算が必要
    - ハードウェアでの除算はかなり負荷が高い

射影テクスチャについて (3)

- ハードウェア設計者の観点から見たテクスチャ
  - パースペクティブ補正テクスチャは実際的な要件
    - そうでないとテクスチャが “泳ぐ”
    - パースペクティブ補正テクスチャでは既にフラグメント単位除算のハードウェア コストがかかっている
  - 賢明なアイディア **[Segal, et.al. ‘92]**
    - 単に **1/w** ではなく **q/w** を補間
    - すると、既にパースペクティブ補正テクスチャ生成を行っている場合、実質的に射影テクスチャにはコストがかからない

射影テクスチャについて (4)

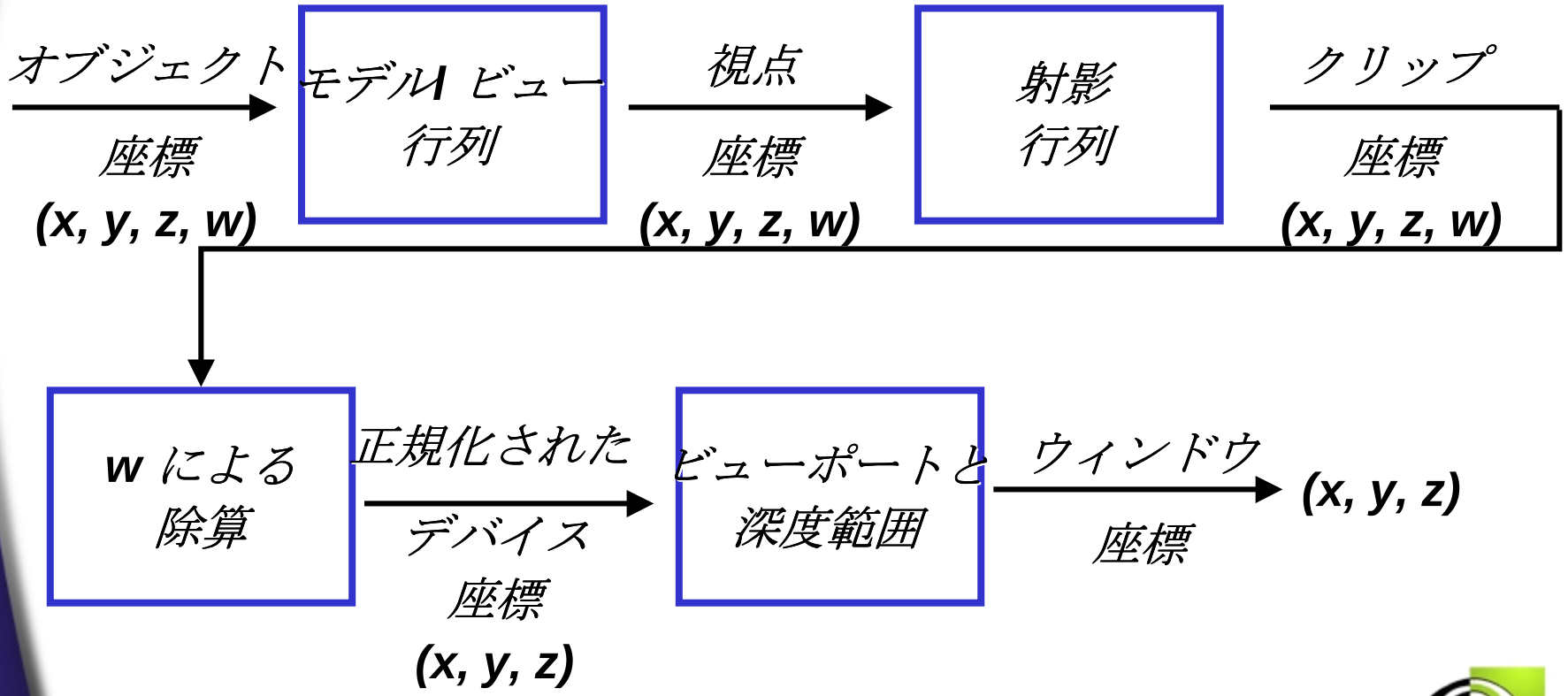
- ハードウェアをだまして射影テクスチャ生成を行うには
  - **q/w** の補間によってハードウェアはフラグメント単位で計算
    - **(s/w) / (q/w) = s/q**
    - **(t/w) / (q/w) = t/q**
  - 最終結果： 射影テクスチャ
    - **OpenGL** は射影テクスチャ生成を仕様化している
    - 唯一のオーバーヘッドは **q** による **1/w** の乗算
    - しかしこれは頂点単位

再びシャドウ マッピングの話題

- **texgen** でライト空間テクスチャ座標を割り当てる
  - 視点空間 **(x, y, z, w)** 座標をライトの視錐台に座標変換 **(**ライトの深度マップ生成方法に一致**)**
  - これらの座標をさらに変換して、ライトのビューの深度マップに直接マッピングする
  - 射影変換として表現できる
    - この変換を、**S**、**T**、および **Q** 座標の **4** つの視点線形平面式に読み込む
  - **(s/q, t/q)** はライトの深度マップ テクスチャに対応

OpenGL の標準的な頂点座標変換

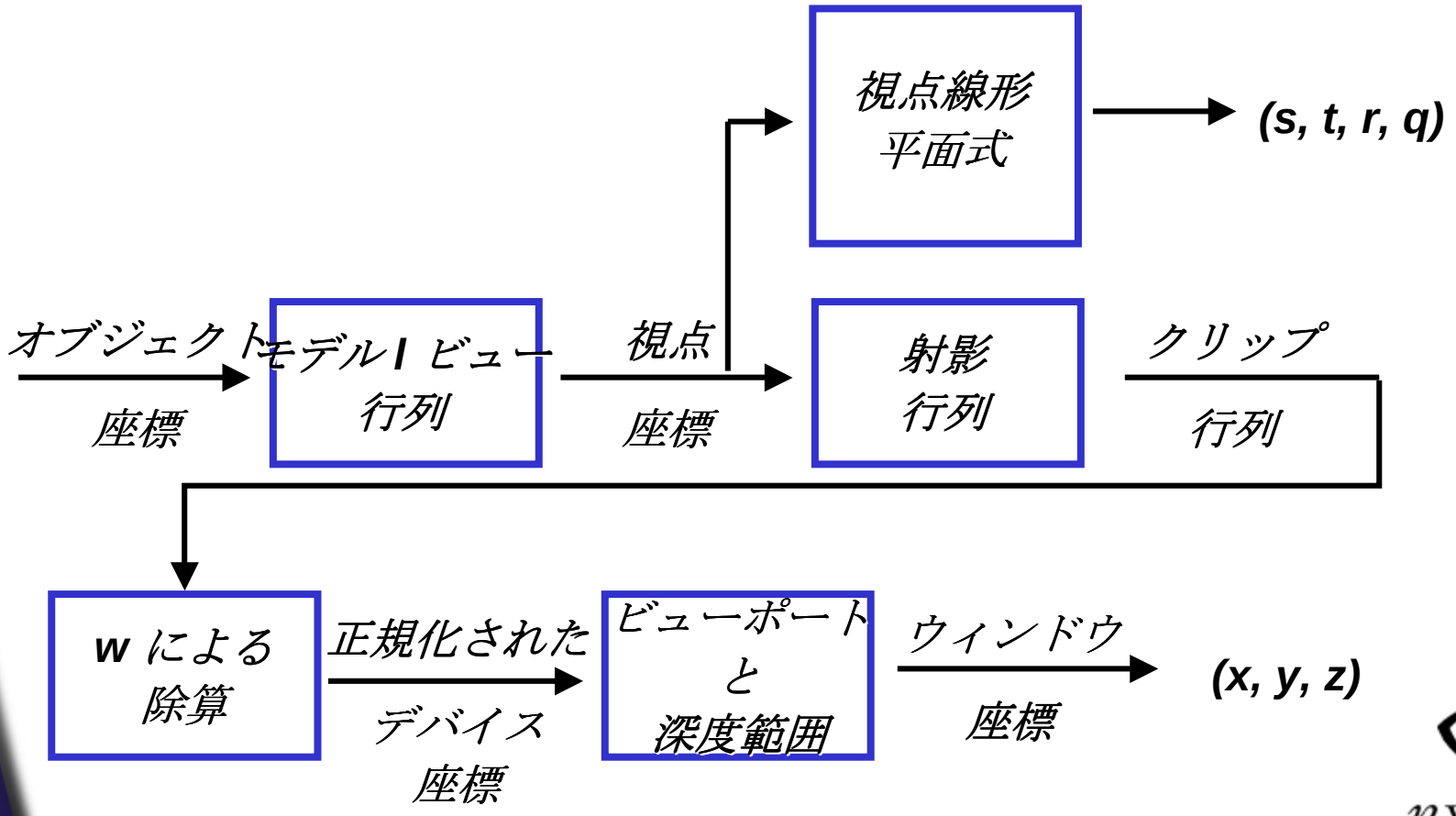
- オブジェクト座標からウィンドウ座標へ



NVIDIA.

視点線形テクスチャ座標の生成

- 視点空間からテクスチャ座標を生成



視点線形 **Texgen** のセットアップ

- **OpenGL** の場合

```
GLfloat Splane[4], Tplane[4], Rplane[4], Qplane[4];
glTexGenfv(GL_S, GL_EYE_PLANE, Splane);
glTexGenfv(GL_T, GL_EYE_PLANE, Tplane);
glTexGenfv(GL_R, GL_EYE_PLANE, Rplane);
glTexGenfv(GL_Q, GL_EYE_PLANE, Qplane);
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
glEnable(GL_TEXTURE_GEN_R);
glEnable(GL_TEXTURE_GEN_Q);
```

- 各視点平面式を現在の逆モデルビュー行列によって変換 **(**我々には非常に便利な方法**)**
  - 注 **: texgen** オブジェクト平面は逆モデルビューによって変換されない

視点線形 **Texgen** 変換

- 平面式が射影変換を構成

$$\begin{bmatrix} s \\ t \\ r \\ q \end{bmatrix} = \begin{bmatrix} Splane[0] & Splane[1] & Splane[2] & Splane[3] \\ Tplane[0] & Tplane[1] & Tplane[2] & Tplane[3] \\ Rplane[0] & Rplane[1] & Rplane[2] & Rplane[3] \\ Qplane[0] & Qplane[1] & Qplane[2] & Qplane[3] \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ w_e \end{bmatrix}$$

- **4** つの視点線形平面式が **4x4** 行列を構成 **(**テクスチャ行列は不要**)**

シャドウ マップ視点線形
Texgen 変換

$$\begin{bmatrix} x_e \\ y_e \\ z_e \\ w_e \end{bmatrix} = \begin{bmatrix} \text{視点} \\ \text{ビュー} \\ \text{(look at)} \\ \text{行列} \end{bmatrix} \begin{bmatrix} \text{モデリング} \\ \text{行列} \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \\ w_o \end{bmatrix}$$

モデル ビュー行列に視点ビュー 変換のみが含まれている場合、glTexGen は自動的にこれを適用

$$\begin{bmatrix} s \\ t \\ r \\ q \end{bmatrix} = \underbrace{\begin{bmatrix} 1/2 & & & 1/2 \\ & 1/2 & & 1/2 \\ & & 1/2 & 1/2 \\ & & & 1 \end{bmatrix} \begin{bmatrix} \text{視錐台} \\ \text{(射影)} \\ \text{行列} \end{bmatrix} \begin{bmatrix} \text{ライト} \\ \text{ビュー} \\ \text{(look at)} \\ \text{行列} \end{bmatrix}} \begin{bmatrix} \text{逆} \\ \text{視点} \\ \text{ビュー} \\ \text{(look at)} \\ \text{行列} \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ w_e \end{bmatrix}$$

この結合変換を glTexGen に指定

シャドウ マップ操作

- 深度マップの自動ルックアップ
  - 適切な座標変換を行う視点線形 **texgen** が読み込まれた後
    - **(s/q, t/q)** は、フラグメントに対応したライトの深度テクスチャ内の位置
    - **r/q** は、ライトの錐台に対して相対的な、フラグメントの **Z** 平面距離 **([0,1]** の範囲にスケーリングされバイアスされたもの**)**
  - 次に **(s/q, t/q)** のテクスチャ値を値 **r/q** と比較
    - **texture[s/q, t/q] ≅ r/q** ならば、シャドウにならない
    - **texture[s/q, t/q] < r/q** ならば、シャドウになる

専用ハードウェアによるシャドウ マッピングのサポート

- **SGI RealityEngine**、**InfiniteReality**、**GeForce3** および **Xbox** ハードウェア
  - テクスチャ フィルタリング操作としてシャドウ テストを実行
    - **2D** テクスチャの **(s/q, t/q)** でテクセルをルックアップ
    - ルックアップ値を **r/q** と比較
    - テクセルが **r/q** と等しいかそれより大きい場合は、**1.0** を生成
    - テクセルが **r/q** より小さい場合は、**0.0** を生成
  - 結果によって色を調整
    - フラグメントがシャドウになる場合はゼロ、ならない場合は色は変更しない

シャドウ マップ ハードウェアに対する OpenGL の拡張

- **2** つの拡張機能の連携動作
  - **SGIX_depth_texture**
    - 高精度の深度テクスチャ フォーマットをサポート
    - 深度バッファからテクスチャ メモリへのコピーがサポートされる
  - **SGIX_shadow**
    - **“**シャドウ比較**”** テクスチャ フィルタリング モードを追加
    - **r/q** を **(s/q, t/q)** のテクセル値と比較
  - マルチベンダ サポート**: SGI**、**NVIDIA**、その他**?**
    - **Brian Paul** 氏によってこれらの拡張機能が **Mesa** に実装された

NVIDIA.

深度テクスチャの新しい内部テクスチャ フォーマット

- **SGIX_depth_texture** はシャドウ マッピング用の深度値を含むテクスチャをサポート
- **3** つの新しい内部フォーマット
  - **GL_DEPTH_COMPONENT16_SGIX**
  - **GL_DEPTH_COMPONENT24_SGIX**
  - **GL_DEPTH_COMPONENT32_SGIX (GeForce3** の **24** ビットと同じ**)**
- 外部フォーマットには **GL_DEPTH_COMPONENT** を使用
- **glCopySubTexImage2D** と連携して動作し、深度バッファからテクスチャへの高速コピーを実現
  - **NVIDIA** はこれらのコピー テクスチャ パスを最適化

深度テクスチャの詳細

- 使用例：

```
glCopyTexImage2D(GL_TEXTURE_2D, level=0,
   internalfmt=GL_DEPTH_COMPONENT24_SGIX,
   x=0, y=0, w=256, h=256, border=0);
```

- テクスチャの最初の内部フォーマットを定義した後は、**glCopySubTexImage2D** の使用によって更新がより高速になる
- ヒント：テクスチャ内部フォーマットには **GL_DEPTH_COMPONENT** を使うこと
  - **“n_SGIX”** の精度指定を使用しない場合、ドライバは深度バッファの精度に合わせる
  - 深度バッファの精度が深度テクスチャの精度に一致している場合に、テクスチャのコピーのパフォーマンスは最大になる

深度テクスチャ コピーのパフォーマンス

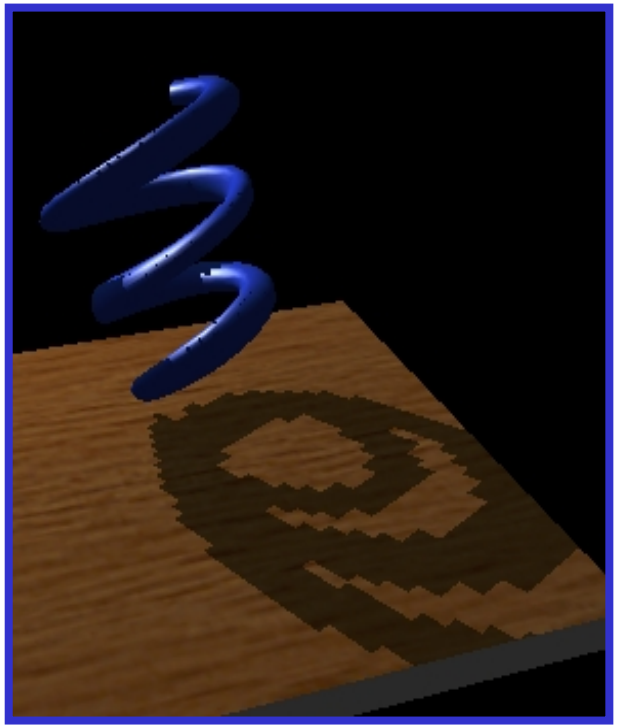
- コピーする深度値が多いほどパフォーマンスは低下
  - **512x512** のコピーは **256x256** の **4** 倍長くかかる
  - トレードオフ：より良いシャドウには、より高い解像度のシャドウ マップが必要だが、コピー速度は低下
- **16** ビット深度値は **24** ビット深度値 **(32** ビット ワード内に含まれている**)** よりもコピーが **2** 倍高速
  - **16** ビット深度バッファを要求し **(32** ビット カラー バッファがあっても**)**、**16** ビット深度テクスチャにコピーすると、**24** ビット深度バッファを使うよりも高速
  - **16** ビット深度バッファを使用すると通常はステンシルが使えないことに注意

ハードウェアのシャドウ マップ フィルタリング

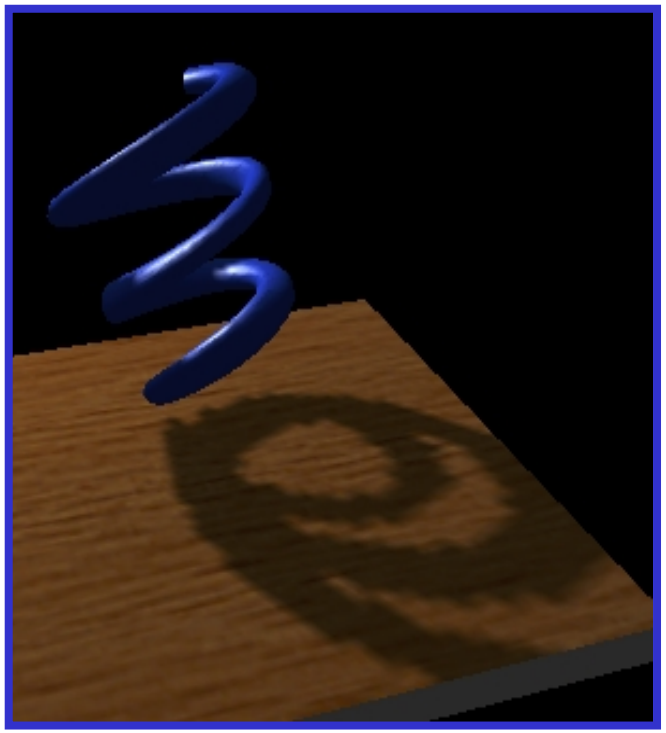
- **“Percentage Closer”** フィルタリング
  - 通常のテクスチャ フィルタリングは色成分を平均化するだけ
  - 深度値の平均化は機能しない
  - 解決策 **[Reeves, SIGGARPH 87]**
    - ハードウェアでサンプルごとに比較を実行
    - 次に、比較の結果を平均化する
  - シャドウ マップ エッジにアンチエイリアシングを提供
    - 本影 / 半影の意味でのソフト シャドウではない

ハードウェアのシャドウ マップ フィルタリングの例

GL_NEAREST: むらがある



GL_LINEAR: アンチエイリアス処理されたエッジ

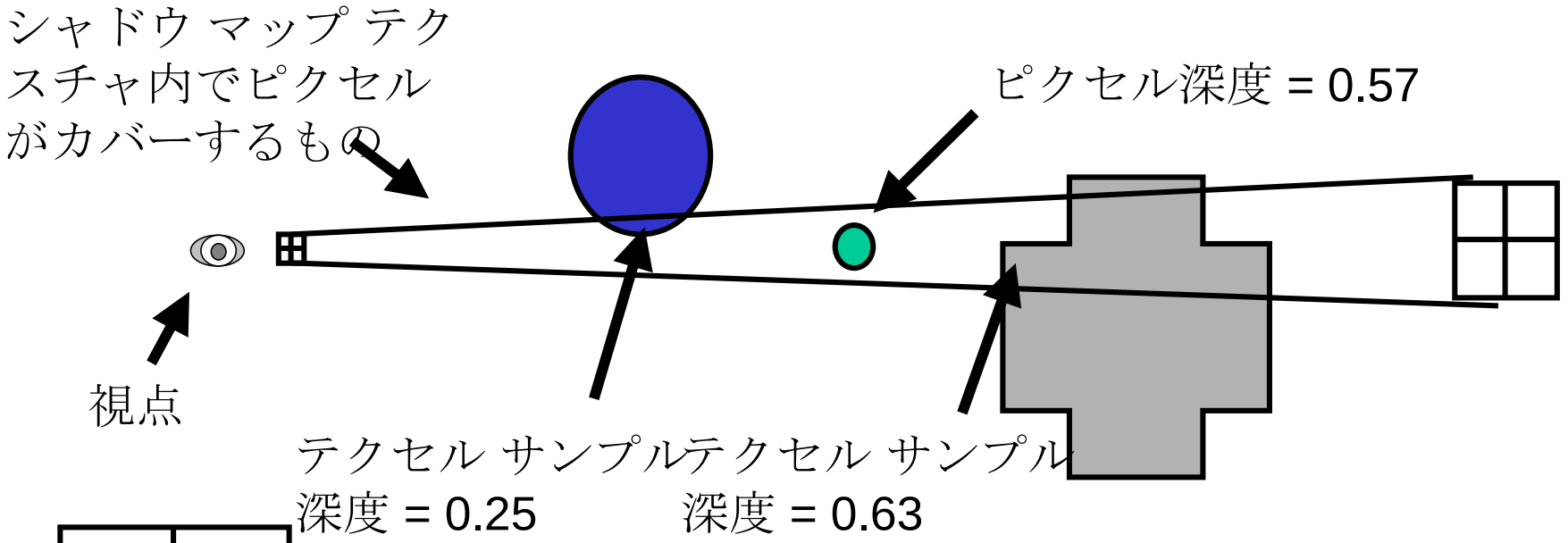


フィルタリングの不自然な効果を強調するために低いシャドウ マップ解像度を使用している

NVIDIA.

深度値はブレンド不可能

- 従来のフィルタリングは不適切



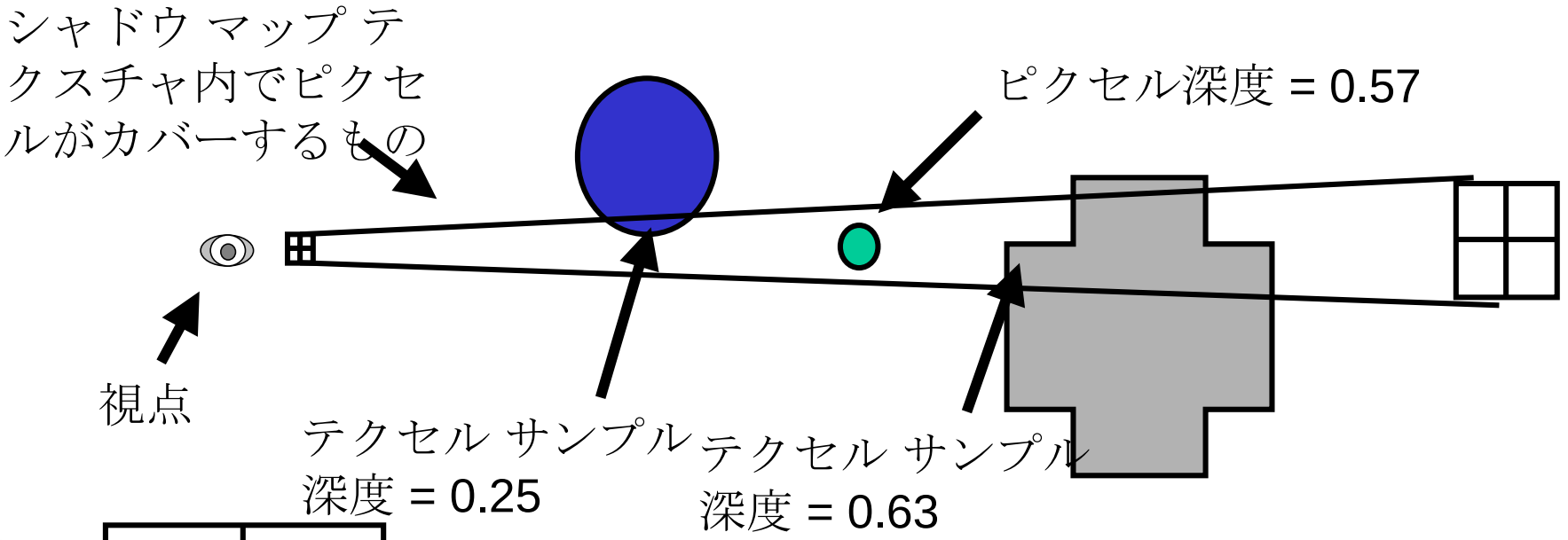
0.25	0.25
0.63	0.63

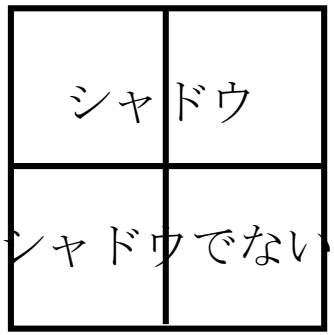
Average(0.25, 0.25, 0.63, 0.63) = 0.44
0.57 > 0.44 であるのでピクセルが “シャドウの中にある” のは誤り
正解 : 0.44 には何もない。0.25 と 0.57 のみ

NVIDIA

Percentage Closer フィルタリング

- 深度値ではなく比較の*結果*を平均化





Average(0.57>0.25, 0.57>0.25, 0.57<0.63, 0.57<0.63) = 50%
したがって、ピクセルは当然 **50%** シャドウになる
(実際にはハードウェアは重み付き平均を実行)

NVIDIA.

Percentage Closer フィルタリングを使用した深度テクスチャのミップマップ フィルタリング (1)

- ミップマップ フィルタリングが機能する
  - サンプリングした **1** つまたは **2** つのミップマップ レベルの比較結果を平均化
- **gluBuild2DMipmaps** を使用して深度テクスチャ ミップマップを作成することはできない
  - 深度値はブレンドできないため
- ミップマップが必要な場合、必要なそれぞれの解像度でシーンをレンダリングし直すのが最適なアプローチ
  - 通常、すべてのミップマップ レベルでこれを行うのはコストがかかりすぎて現実的でない
  - **OpenGL 1.2** の **LOD** 固定は、 **1x1** レベルまでの全てのレンダリングを回避するのに役立つ

Percentage Closer フィルタリングを使用した深度テクスチャのミップマップ フィルタリング (2)

- ミップマップにより、セルフ シャドウの回避を保証する適切なポリゴン オフセット スケールおよびバイアスの特定が難しくなる場合がある
- たとえば **512x512** および **256x256** という **2** つのミップマップ レベルを使用し、最小および最大 **LOD** 固定を **0.5** に設定することで、**“8-tap”** フィルタリングができる
  - **OpenGL 1.2** の **LOD** 固定を使用

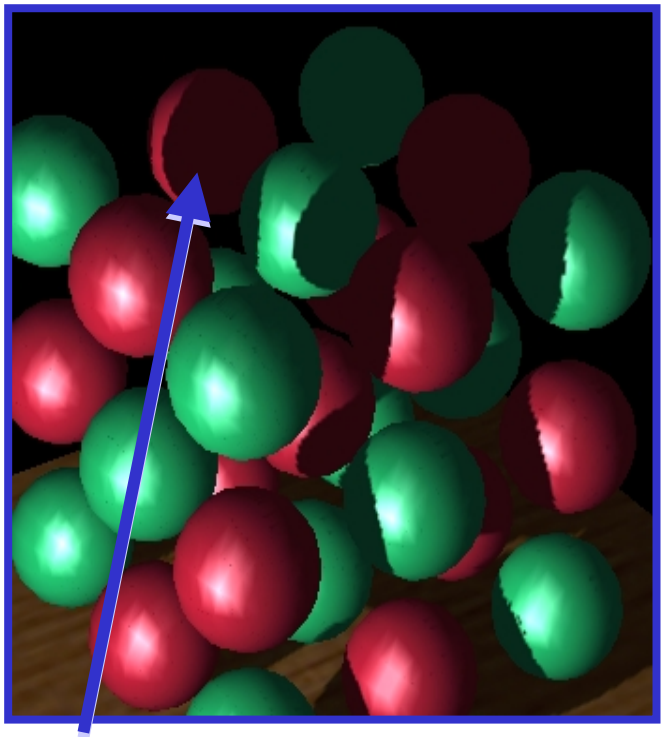
シャドウを使用した照明モデルに関するアドバイス **(1)**

- デカール テクスチャを使用した一般的な照明モデル：
  *(ambient + diffuse) * decal + specular*
- シャドウ マップはシャドウ項を提供
- シャドウ マップがシャドウ項 ***shade*** を提供すると想定
  - シャドウのパーセンテージ
  - **100% =** シャドウなしで表示、**0% =** 完全にシャドウ
- シャドウイング用に改定した照明モデル：
  *(ambient + shade * diffuse) * decal + shade * specular*
- 問題は現実世界の光源の場合、シャドウになる表面上にディフューズ シェーディングを **100%** さえぎることはないということ
  - 光は拡散する。現実世界の光は理想的な点ではない

NVIDIA.

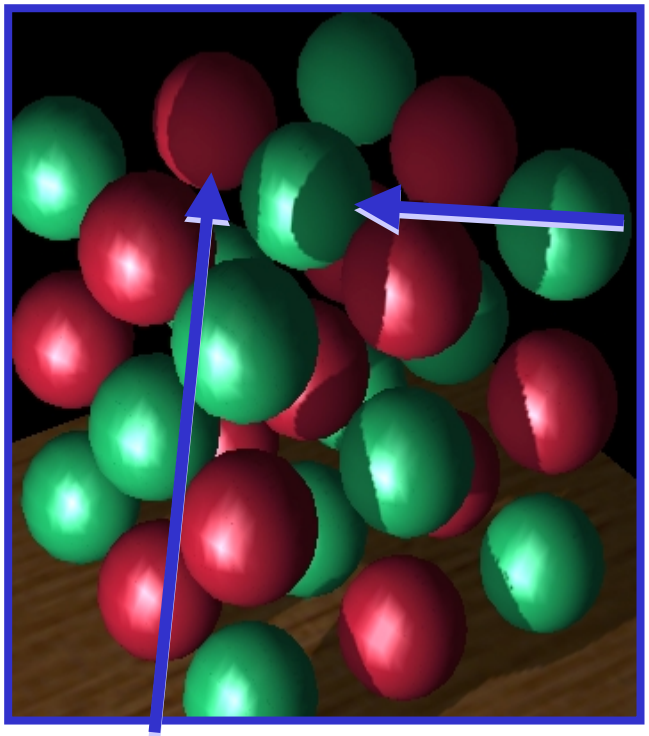
薄暗いディフューズ (Dimming) にする必要がある

Dimming なし：シャドウの領域は
0% のディフューズ、
0% のスペキュラ



前面に向いているシャドウの領域が不自然にフラットに見える

Dimming あり：シャドウの領域は
40% のディフューズ、
0% のスペキュラ



シャドウの領域に湾曲が見える

どちらの場合もシャドウの領域にスペキュラはない

シャドウを使用した照明モデルに関するアドバイス **(2)**

- **Dimming** した照明モデル **:**

  *(ambient + diffuseShade * diffuse) * decal + specular * shade*

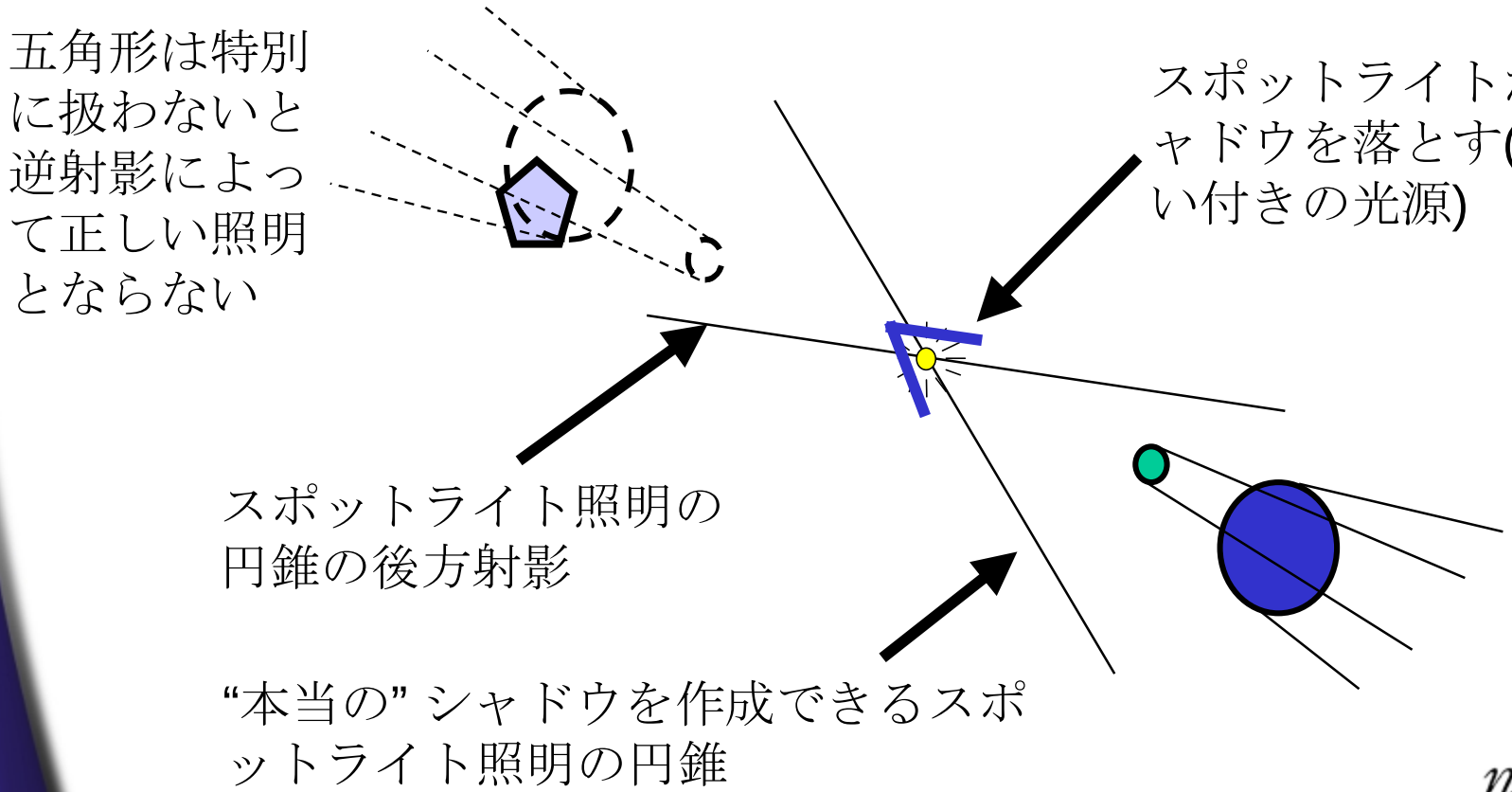
  diffuseShade は次のとおり

  *diffuseShade = dimming + (1.0 – dimming) * shade*

- **NV_register_combiner** と **OpenGL 1.2** の **“**分離スペキュラ カラー**”** サポートを使用して、簡単に実装可能
  - 分離スペキュラにより、ディフューズおよびスペキュラの頂点単位ライティング結果が別々に維持される
  - NV_register_combiner により、プライマリ (ディフューズ) カラーとセカンダリ (スペキュラ) カラーをピクセル単位で上記の算術式と組み合わせることができる

後方射影シャドウ マップについての注意 (1)

- 標準射影テクスチャと同じように、シャドウ マップも後方射影が発生し得る



NVIDIA

後方射影シャドウ マップについての注意 (2)

- 後方射影を排除するテクニック：
  - シャドウ マップ結果を、適切なカットオフを持つ 1 つの頂点単位スポットライトのライティング結果で調整する (そのスポットライトの後方ライトを必ず “オフ” にすること)
  - ライトからの平面距離が “s” である小さい 1D テクスチャを使用すると (“s” は平面 texgen モードで生成)、1D テクスチャは負の距離については 0.0、正の距離については 1.0 になる
  - クリップ平面を使用、これにはライトの位置とスポットライトの方向によって定義される平面を用いる
  - シャドウ マップを適用するとき、ライトの “後ろ” にジオメトリを作成することは避ける (クリップ平面よりもよい)
  - NV_texture_shader の GL_PASS_THROUGH_NV モード

シャドウ マッピングの改善に役立つその他の OpenGL 拡張機能

- **ARB_pbuffer** – シャドウ マップ深度バッファをレンダリングするためのオフスクリーン レンダリング サーフェスを作成
  - 通常はバック バッファにシャドウ マップを作成し、それをテクスチャにコピーできる
  - しかし、シャドウ マップの解像度がウィンドウの解像度よりも高い場合は pbuffer を使用すること
- **NV_texture_rectangle** – テクスチャの幅および高さが 2 の累乗でなくてもよい新しい 2D テクスチャ ターゲット
  - 制限
    - ミップマップまたはミップマップ フィルタリングはサポートされていない
    - ラップ固定モードがない
    - テクスチャ座標の範囲は [0..1]x[0..1] ではなく [0..w]x[0..h]
  - シャドウ マッピングに十分適す

シャドウ マッピングと その他のテクニックの組み合わせ

- 組み合わせてうまく機能するテクニック
  - ステンシルを使用して、シャドウの内側か外側かを示すタグをピクセルに付ける
  - 追加のパスでほかのレンダリング テクニックを使用
    - バンプ マッピング
    - テクスチャ デカールなど
  - シャドウ マッピングはより複雑なマルチパス レンダリング アルゴリズムと統合可能
- シャドウ マッピング アルゴリズムでは頂点レベル データへのアクセスは不要
  - 頂点プログラムなどとの混在が簡単

シャドウ マッピング機能を持つハードウェア以外の手段

- コンシューマ向け **3D** ハードウェア ソリューション
  - **Wolfgang Heidrich** が **1999** 年の博士論文で提唱
  - 現行のコンシューマ向けマルチ テクスチャ ハードウェアを活用
    - 第 **1** テクスチャ ユニットは **2D** 深度マップ テクスチャにアクセス
    - 第 **2** テクスチャ ユニットは **1D Z** 範囲テクスチャにアクセス
  - 拡張されたテクスチャ環境では、第 **1** テクスチャから第 **2** テクスチャを減算
    - **0** より大きい場合はシャドウになり、そうでない場合はシャドウにならない
    - シャドウになったフラグメントを破棄するには、アルファ テストを使用

シャドウ マッピングの課題 (1)

- 問題はある
  - 不自然なエイリアシングが発生しやすい
    - **“Percentage Closer”** フィルタリングで改善できる
    - 通常のカラー フィルタリングはうまく機能しない
  - 深度バイアスは誰にでも使いやすいとは言えない
  - 追加のシャドウ マップ レンダリング パスとテクスチャ読み込みが必要
  - 高解像度のシャドウ マップを使えばムラは減る
    - しかし、テクスチャをコピーするコストも増える

シャドウ マッピングの課題 (2)

- 問題はある
  - シャドウは視錐台の数に限定される
    - 全方向ライト用に **6** つの視錐台を使用可能
  - 近クリップ面、遠クリップ面の外側にある、またはクリップ面と交差するオブジェクトは、シャドウイングで正しく処理されない
    - 近クリップ面はできるだけ近くに移動する
    - ただし、近づけすぎると、射影錐台を使う際に高価な深度マップ精度が無駄になる

シャドウ マップの解像度を決めるときの原則 (1)

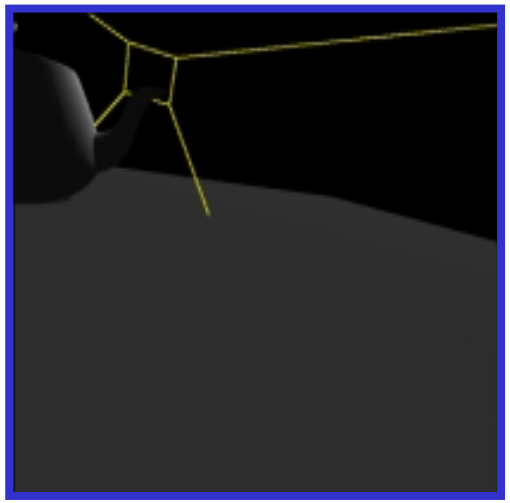
- ライトのビューのピクセル (サンプル) と、視線のビューのピクセル (サンプル) とのサイズの対応がわかっている必要がある
  - 再サンプリングの問題
- 光源錐台と視点錐台が妥当な範囲で正しく調節されている場合、サンプル サイズの割合は **1.0** に近くなる
  - 視線とライトの錐台がほぼ同一であれば正しい対応
  - しかし、それは可視シャドウがほとんどないことを示唆する
  - 坑夫のランプを考える (ヘルメットに付いているライトなど)
  - このようなランプを使う主な理由は、ランプを装着している間はランプのシャドウが見えないということ

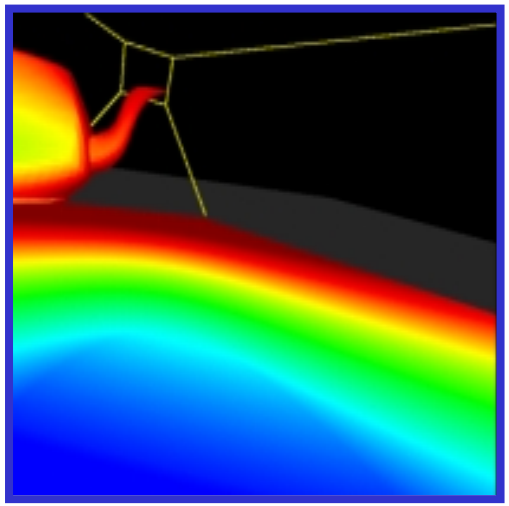
シャドウ マップの解像度を決めるときの原則 (2)

- 最適なケースは坑夫のランプ
- 最悪なケースは見る人を照らしているライトのシャドウ
  - “ヘッドライトに照らされた鹿” の問題 – 鹿にとっては明らかに最悪のケース
  - “錐台の決闘” 問題としても知られている
- このことを可視化した例を示す...

錐台の決闘ケースのイメージ

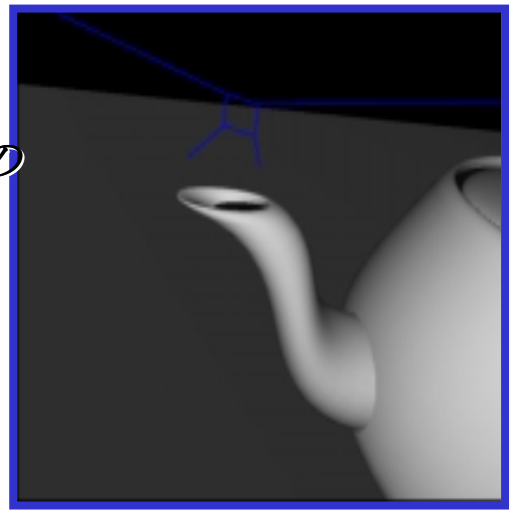
視点のビュー

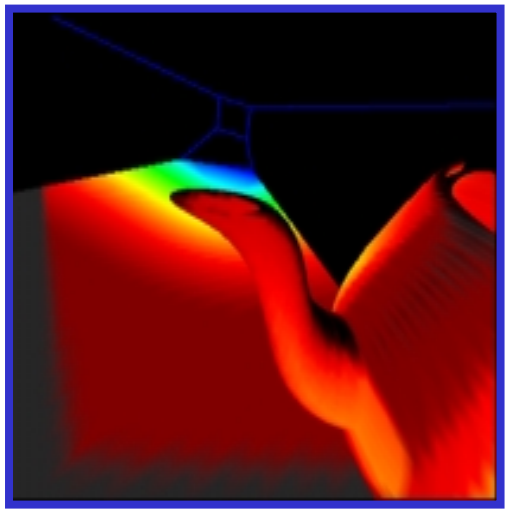




ライトのカラー コード化されたミップマップ レベルの射影を使用した視点のビュー:
青 = 最大
赤 = 最小

ライトのビュー



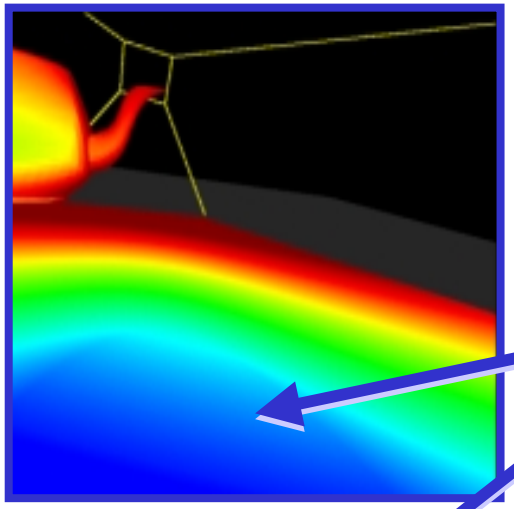


上記の視点からのイメージを再射影したライトのビュー

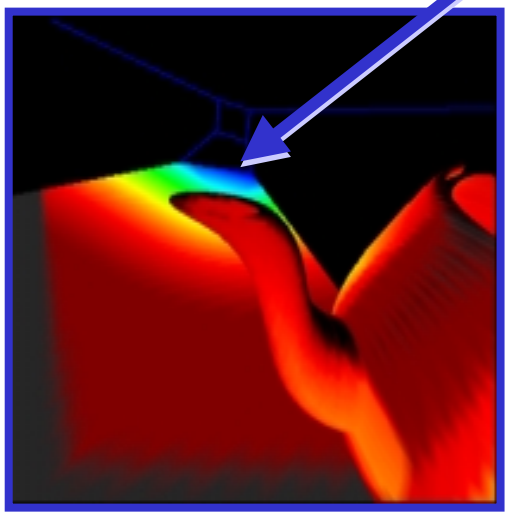
NVIDIA

錐台の決闘ケースにおける 4 つのイメージの解釈

視点のビュー



ライトのビュー

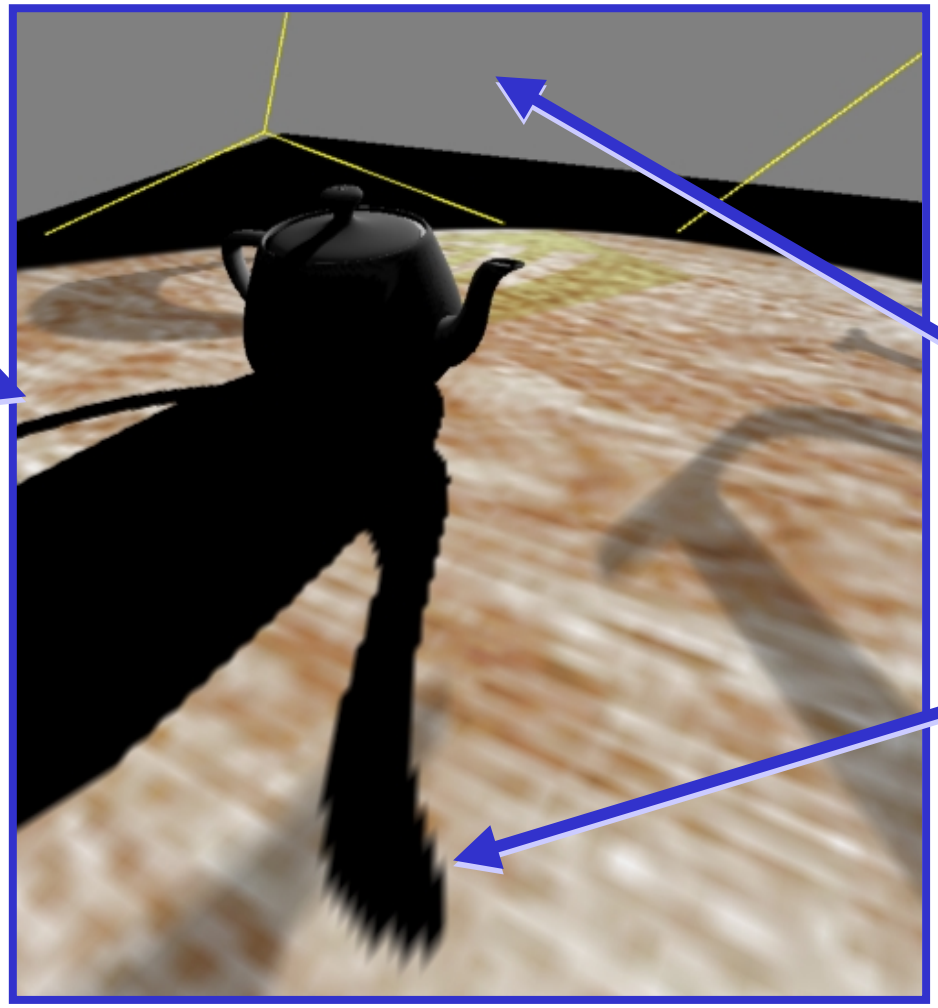


ライトのビューの最小領域は、視点のビューでは非常に大きい領域になっている。これは、明らかなムラがあるシャドウ エッジによる不自然な効果を避けるためには、非常に高解像度のシャドウ マップが必要になることを意味する。

NVIDIA.

錐台の決闘状況における シャドウ エッジの不自然な効果 (ムラ) の例

シャドウ エッジがこの距離では正しく定義されている点に注意



この位置では、ライトは見る人に向かっている

ムラがあるシャドウ エッジの不自然な効果

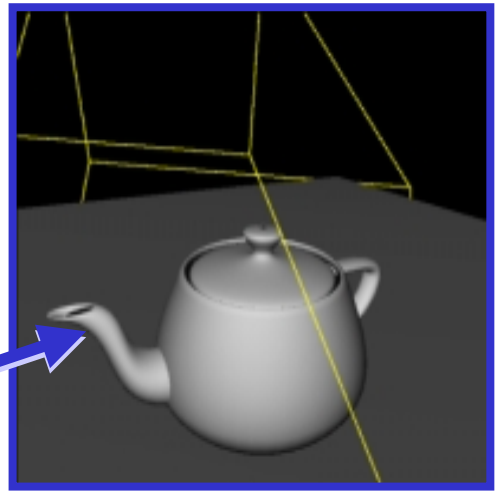
NVIDIA

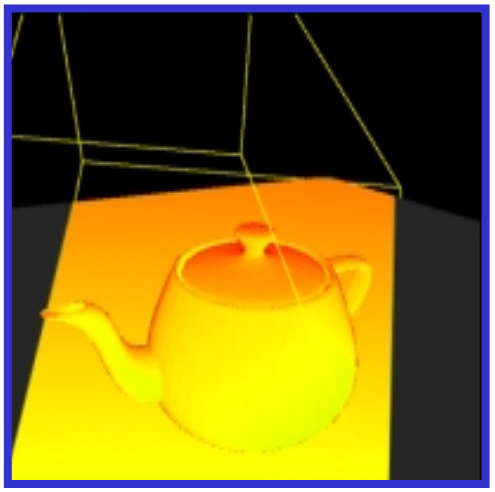
良い例 (坑夫のランプに近い)

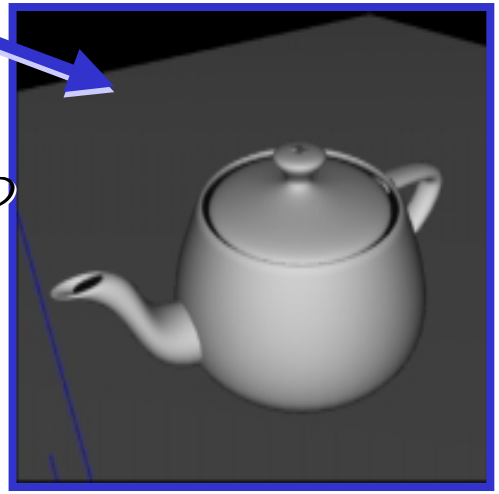
視点のビュー

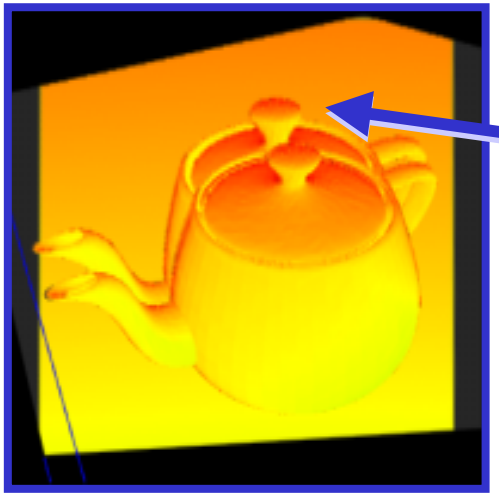
よく似たビューー

ライトのビュー









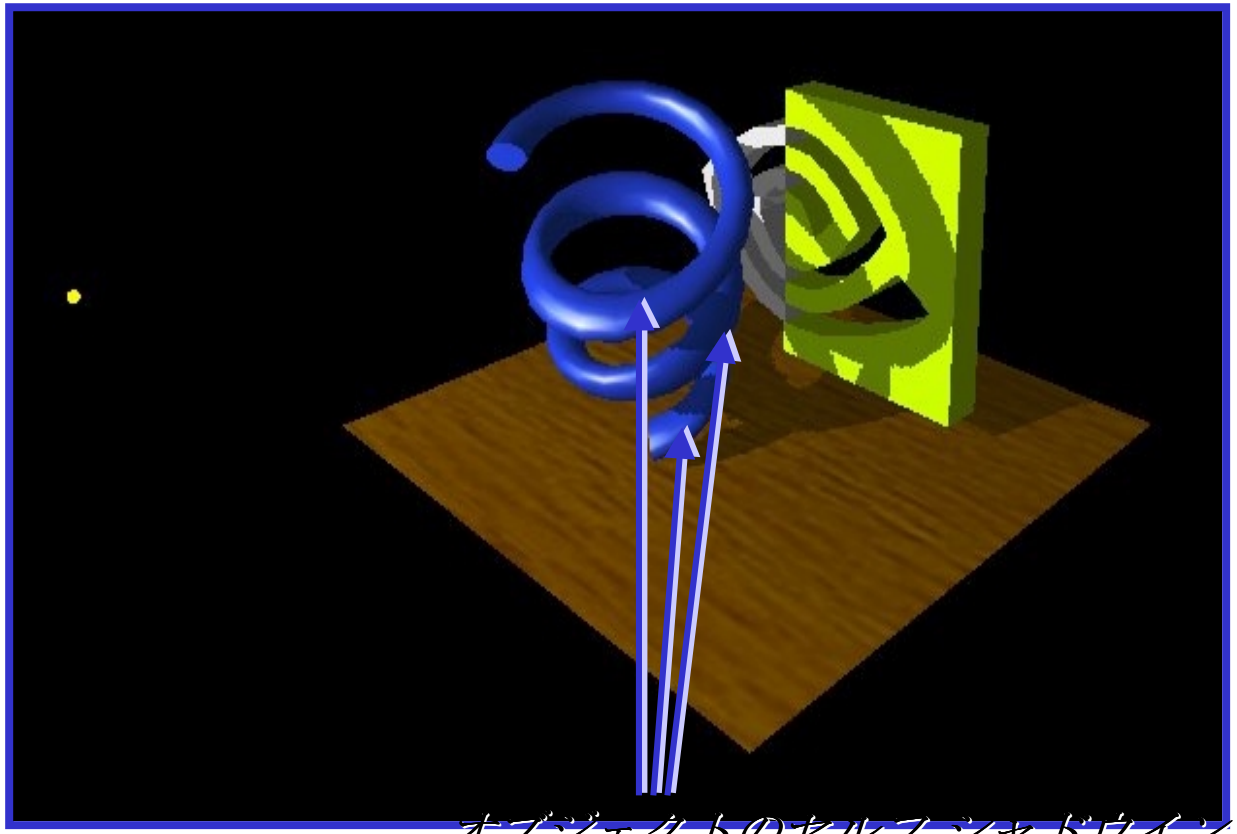
カラー コード化されたイメージが類似したパターンを共有しており、統一性のある色付けがされている点に注意。単一の深度マップ解像度がシーンの大部分に対してうまく機能していることを意味する

ゴーストは射影がシャドウに入る部分に発生する

NVIDIA.

その他の例

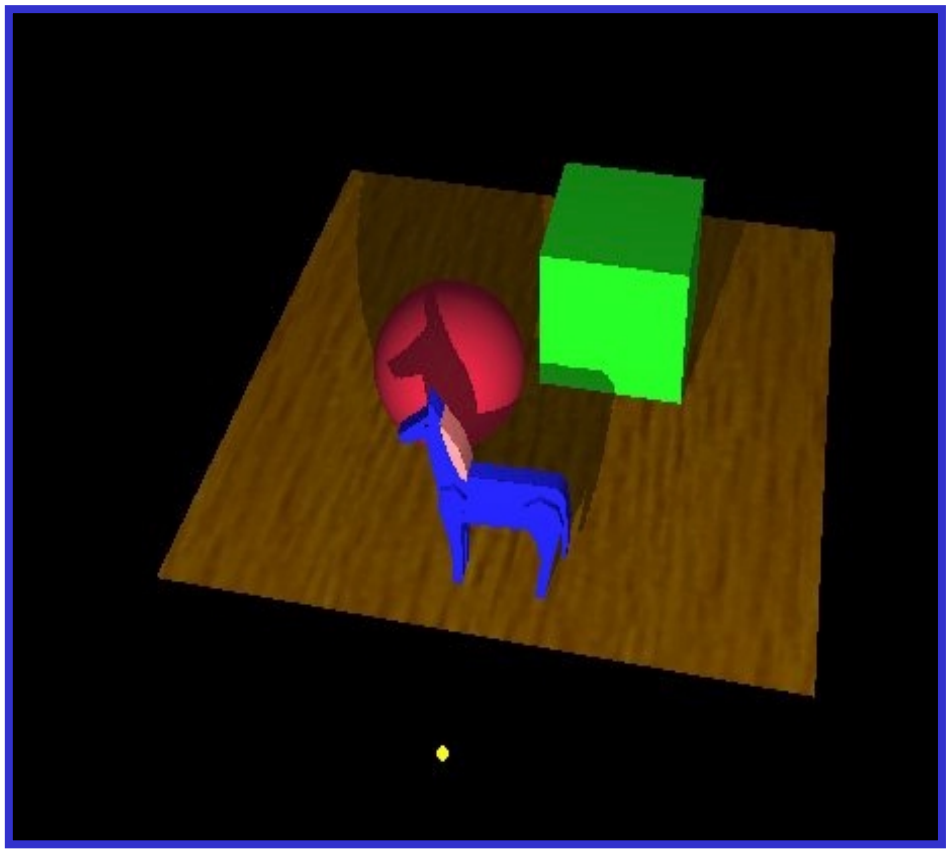
- オブジェクトのセルフ シャドウイングによるスムーズなサーフェス



オブジェクトのセルフ シャドウイングに注意

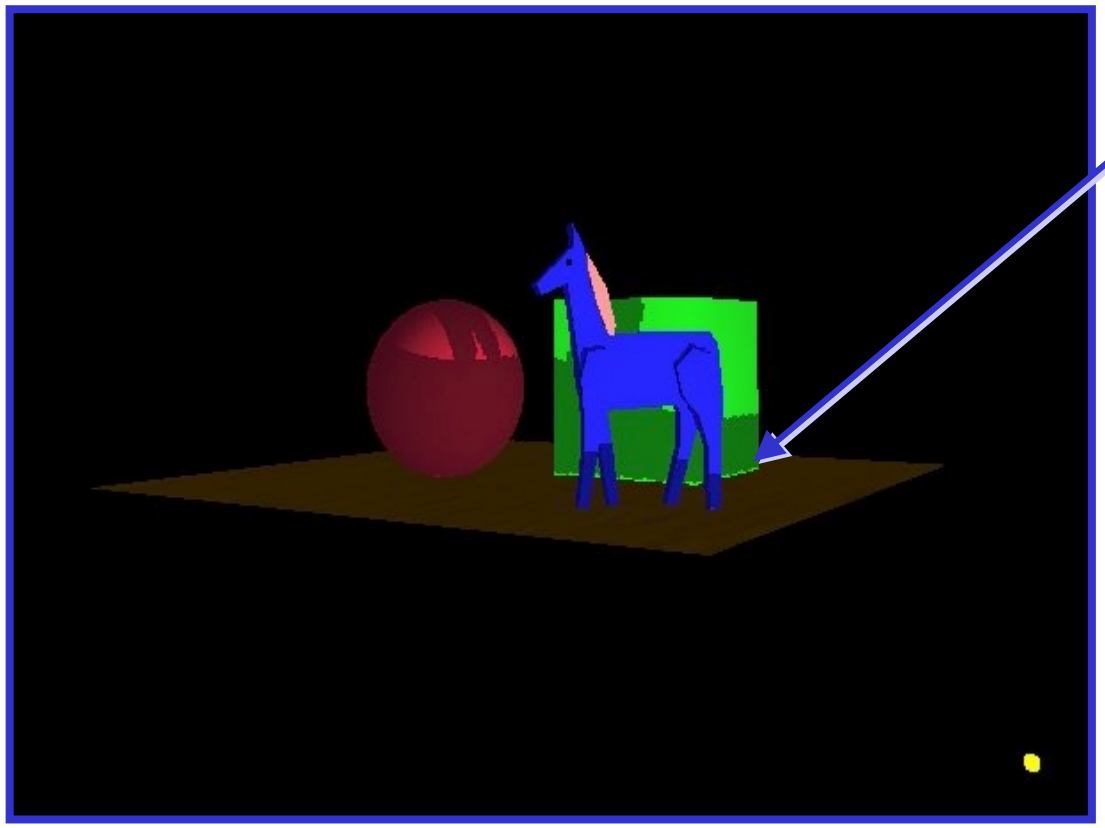
その他の例

- 複雑なオブジェクトが全てシャドウを持つ



その他の例

- フロアもシャドウを落とす

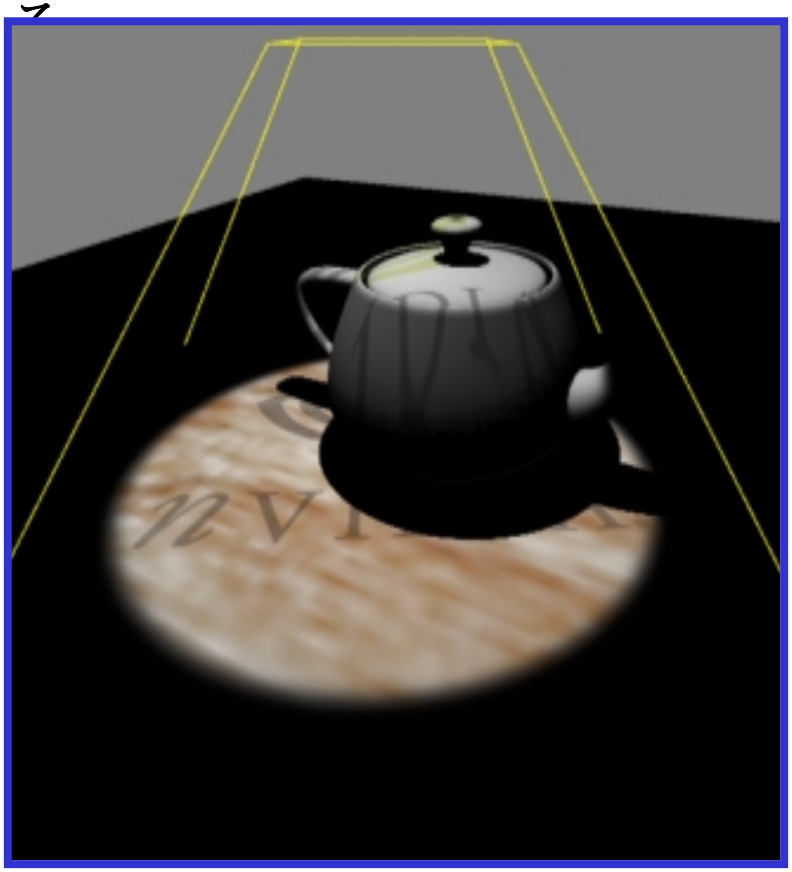


フロアが無限に薄いためシャドウが漏れている点に注意

フロアを厚くすることで修正可能

スポットライト シャドウの射影テクスチャとの組み合わせ

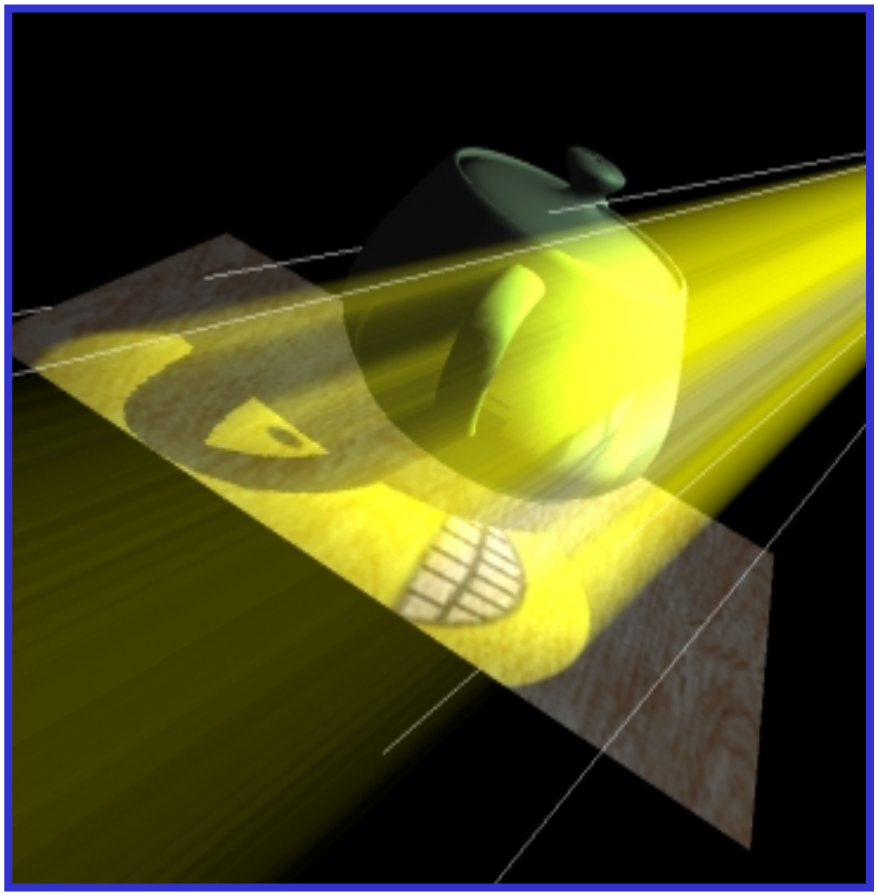
- スポットライト形式の射影テクスチャを使用してシャドウ マップにスポットライト フォールオフを当てる



NVIDIA

シャドウと環境エフェクトとの組み合わせ

- ほこりっぽい部屋のシャドウ



浮遊しているほこりなどの環境エフェクトをシミュレート

1) *シャドウ マップを作成*

2) *シャドウ マップを使用してシーンを作成*

3) *射影テクスチャ イメージと射影シャドウ マップを調整*

4) *背面から前面へのシャドウ スライス平面 (これも射影テクスチャ イメージで調整したもの) をブレンド*

提供：**Cass Everitt**

NVIDIA.

シャドウ マッピングを使用したリアルタイム **Luxo Jr.**

- **MacWorld Japan** での **Steve Jobs** のデモンストレーション **(Mac** 上の **OpenGL** でハードウェア シャドウ マッピングを使用**)**

NVIDIA

Luxo Jr. のデモの詳細

- **Luxo Jr.** にはアニメーション化されたライトが **2** つと、天井にあるライトが **1** つ使われている
  - **3** つのシャドウ マップをフレームごとに*動的に*生成
- 複雑なジオメトリ **(**コードとランプのアーム**)** はすべて正しくシャドウ処理されている
- ユーザーがビューをコントロールしても、シャドウイングが機能
- リアルタイムの **Luxo Jr.** は **OpenGL** にとっての技術的な偉業
- **OpenGL** でのみ利用可能

(申し訳ありませんがデモは省略させていただきます。イメージは Apple 社 MacWorld Japan の Web 放送ビデオのものです。)

NVIDIA

シャドウ マッピングのソース コード

- **NVIDIA** の **Web** サイト
  - ソース コード
    - **“**シャドウキャスト**” OpenGL** コード例
    - 利用可能な最高のシャドウ マッピング サポートを使用している **TNT**、**GeForce**、**Quadro**、**GeForce3** で実行可能
    - **EXT_texture_env_combine** をサポートしているベンダでも実行可能
  - ***NVIDIA OpenGL Extension Specifications***
    - **EXT_texture_env_combine**、**NV_register_combiners**、**SGIX_depth_texture**、**SGIX_shadow** についてのドキュメント
  - **http://www.nvidia.com**

クレジット

- ここで示したアイディアの基になった資料
  - **Wolfgang Heidrich, Max-Planck Institute for Computer Science**
    - 最初のデュアル テクスチャ シャドウ マッピングのアイディア
    - 氏の論文『***High-quality Shading and Lighting for Hardware-accelerated Rendering***』
  - **Michael McCool, University of Waterloo**
    - マルチディジットのシャドウ比較のアイディアを提唱

まとめ

- シャドウ マッピングはリアルタイムのシャドウイング エフェクトを実現
  - シーンの複雑さとは無関係
  - マルチテクスチャとの高い適合性
    - ステンシル シャドウ ボリュームとは異なり、マルチパスが必須でない
  - スポットライトのシャドウに最適
- 最新コンシューマ向けハードウェアのシャドウ マップ サポート
  - **GeForce3**
  - デュアルテクスチャ テクニックはレガシー ハードウェアをサポート
- 同じ基本テクニックが **Pixar** の **CG** 映画でシャドウに使用されている