HEXA DRIVE

http://hexadrive.jp/

ベンチャーゲーム開発会社による マルチプラットフォームゲームエンジン開発への挑戦

HEXADRIVEHEXADRIVEHEXADRIVEHEXADRIVEHEXADRIVEHEXADRIVEHEXADRIVE

マルチプラットフォームゲームエンジンの開発…それはベンチャーゲーム開発企業が取り組むには、余りにも遠大な目標です。

ベンチャーゲーム会社は、いつも人手が足りません。ともすれば、各人が自分の作業に追われがちになってしまいます。
その結果、何の対策も講じずにいれば、成果物に対するチェック体制が甘いものになってしまいます。
社外クライアントの方々から、エンジンに関する不具合報告や機能不備を指摘して頂く…という痛い経験を繰り返す度、
我々は、少しずつではありますが成長し、より良い対策を講じて参りました。
今回は、その内の幾つかをご紹介させて頂きます。

デイリービルド & ビルドレポート

いつの間にかMasterビルドでエラーが出るようになってた！ …なんてこと、ありませんか？

ROM 提出間近になって、久しぶりにビルドしてみた Master 構成がエラーだらけ…
なんて経験、ありますよね？
ヘキサドライブでは、そんな事態にならないよう、
全ビルド構成（２１プロジェクト）を、毎日自動ビルドするようにしています。
そしてその結果を各自に自動配信することにより、
担当者が直ちにエラーに対処できる仕組みになっています。
これらの自動ビルド、自動配信はサーバ上で
Python（オブジェクト指向スクリプト）により制御され、
人手を介することなく運営されています。

```
# バッチビルドバッチ
BATCHBUILD_BAT = 'bin\\hxbuilder\\batchbuild.bat'

# ビルドの報告をするトピック名
TOPIC_NAME = 'エンジン デイリービルド'

RE_PS3_BUILD_RESULT = re.compile('([0-9]*>\s)*.* - エラー')

# ビルド結果表示
def result_message(error_count, warning_count):
    result = '\t エラー %d, 警告 %d' % (error_count, warn
    if error_count == 0:
        result = result + ' : ビルド成功！'
    else:
        result = result + ' : ビルド失敗…'
    return result

# PC 版ビルドレポー
```

スクリプトによる自動ビルド & 自動レポート

```
【ビルドレポート 2010-07-30(Friday) 10:07:17】

■Windows
・Debug   エラー 0, 警告 4 : ビルド成功！
・Master   エラー 0, 警告 8 : ビルド成功！
・Profile   エラー 0, 警告 2 : ビルド成功！
・Profile_FastCap   エラー 0, 警告 2 : ビルド成功！
・Release   エラー 0, 警告 2 : ビルド成功！
・Release_LTCG   エラー 0, 警告 22 : ビルド成功！

■Xbox360
・Debug   エラー 0, 警告 5 : ビルド成功！
・Master   エラー 0, 警告 8 : ビルド成功！
・Profile   エラー 0, 警告 5 : ビルド成功！
・Profile_FastCap   エラー 0, 警告 25 : ビルド成功！
・Release   エラー 0, 警告 5 : ビルド成功！
・Release_LTCG   エラー 0, 警告 12 : ビルド成功！
```

テストプログラム

プログラムの動作確認 自分の手でやってるんですか？

ヘキサドライブでは、テストプログラムによる
自動的な動作確認を取り入れています。
自動テストしにくい描画結果も、
正常動作時のスクリーンショットとピクセル単位で比較することによる、
自動的な動作確認を実現しています！

この他にも、パフォーマンス自動チェック、ソースコードの静的解析による不具合の事前検知、BugTrackingSystemによる不具合管理など
様々な取り組みを行っています。
そういった環境下で制作されたヘキサエンジンに実装されている機能の一部を、以下でご紹介します！

技術紹介

フォワードレンダリングでの高速なSSAO

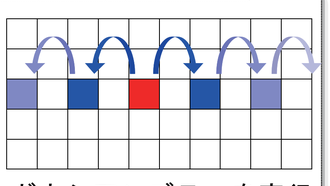
1. 概要
   Deferred Rendering向けのSSAO（Screen Space Ambient Occlusion）を、
   Forward Renderingで実装するための高速化手法の紹介。

2. 実装方法について
   二種類の実装方法。今回は品質面から②を選択。

実装方法	メリット	デメリット
① 深度情報から面の連続性予測	・深度値だけで実装可 ・遮蔽判定が軽量	・遮蔽精度が不十分
② カメラ空間からのレイキャスト	・遮蔽の精度が高い	・計算量が多い ・法線情報が必要

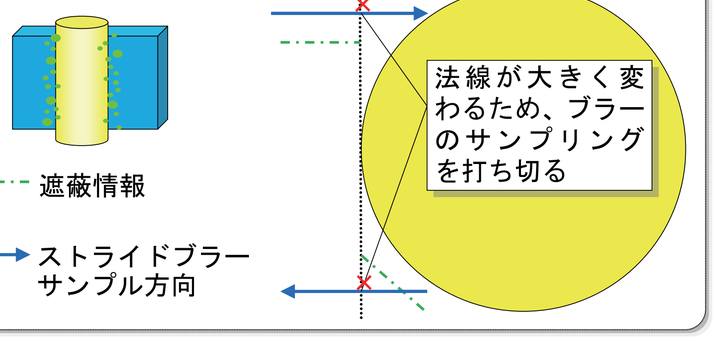
3. 実装と最適化
   シェーダの最適化だけでは限界があったため、バッファサイズの縮小を選択。
   最終的に1/16まで落としたが ”エッジ検出ストライドブラー” で品質を確保。

ストライドブラー



ガウシアンブラーを実行する際、サンプルピクセルを一定数飛ばしてフェッチする。

箱の前に筒状の物体がある状況下での遮蔽



遮蔽情報の生成

深度情報からスクリーンスペースの法線情報を作成

法線情報と深度情報からシーン上の遮蔽を計算

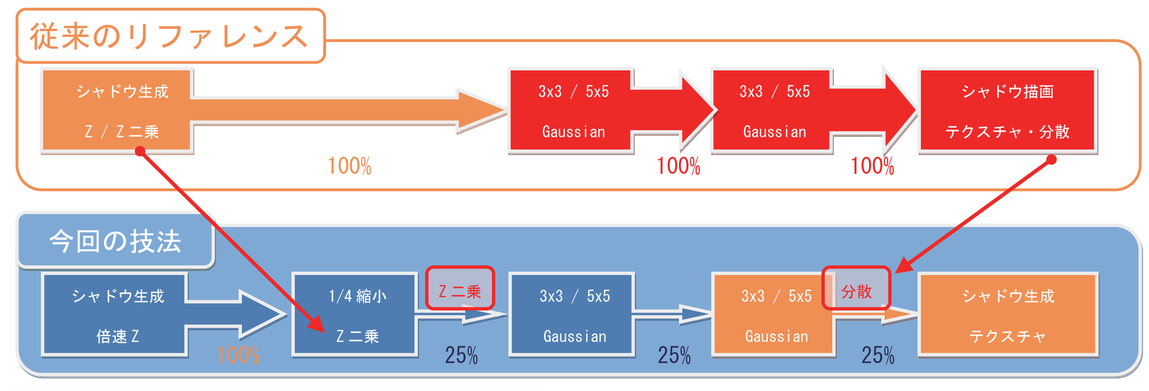
遮蔽情報から高周波ノイズを除去する為にぼかし処理を適用

GPUのメモリ最適化と分散シャドウマップの高速化技法

1. 概要
   ソフトシャドウでは一般的な実装になっている分散シャドウマップ（Variance Shadowmaps）の、GPUの帯域を考慮したフィルタ処理と、VSMの欠点（ライトブリーディングの発生）の解消のための高速な手法を紹介。

2. 従来の一般的な実装との違いについて
   次の点に着目して最適化と式の組み換えを検討。
   (1). シャドウ生成時に倍速Zの恩恵を受けるようにZのみをレンダリング
   (2). ブラーパスのガウシアンフィルタをストライドブラーにする。
   (3). 分散計算を行なう部分の一部を最終パスからブラーパスへ移動して負荷分散
   (4). 最終パスのチェビシェフの不等式を簡略近似で命令サイクル削減

3. 実装と最適化
   PCFとは異なり、VSMで必要となるブラーパスはフィル負荷が問題。
   その対策でバイリニア補間を利用して縮小バッファで処理することでフィル負荷を大幅に削減可能。この場合のエンコードを工夫することで縮小時の情報の損失を低減可能。見た目や精度面で遜色ない低負荷実装。



ソフトシャドウ生成のシェーダー高速化
従来のライトブリーディングの問題の発生を式の組み換えで解決し、なおかつ精度の向上にも効果が上がりました。

従来の実装
```
float d = depth - z;
float light = variance / ( variance + d*d );
```

ライトブリーディング対策込みの今回の手法
```
float light = 1.0 + (z - depth) / sqrt(preVariance + 0.0001 );
```

その他の技術紹介については、こちらへ！

ヘキサドライブ　研究室

http://hexadrive.jp/index.php?id=12