GoForce 3D: あなたのそばのピクセルで

CEDEC 2004

NVIDIAは精力的にハンドヘルドソリューションを開発しています

- エキサイティングで成長中の市場
- 世界で通用するモバイル用グラフィクス製品の開発
- すでに活発な開発が行われている

ハンドヘルド機器のための ゲーム製作?

- 新しい市場
- どこでも使えるモバイル機器
  - 世界で5億以上の端末
  - 30億ドルの呼び出し音ダウンロード
  - 6千万ゲームプレイヤーの内、4千5百万が携帯ゲーム機を使ったことがある
- 気軽なゲーマーと愛好家

ハンドヘルド機器のための ゲーム製作?

- 技術革新でさらなるチャレンジが可能に
  - リアルタイム3D
  - ワイヤレス接続
- 世界がターゲットとなる可能性

チャレンジ...

- システムのリソースが限られている
- 統一されていない開発空間
  - 多様性

… さらにチャレンジ

- 販売経路
- コピーライトの管理
- 独特な市場原理
  - サービス・プロバイダとARPU
  - DATAとモバイル・エンタテイメントへのフォーカス

GoForce 3Dの導入

- ライセンス可能なモバイル機器向けの3Dコア

- OpenGL ESとDirect3Dm準拠
- 低消費電力設計
- 統合されたユニファイドSRAM
- 最大VGAの解像度
- 最新の機能群

- 複雑なゲームを30Hz以上で動作させる設計

GoForce 3D機能

- ジオメトリ・エンジン (floatとfixed)
- 16ビット色と16ビットZ
- 40ビット色(内部表現)
- 最大4つのマルチ・テクスチャ
- バイリニアとトライリニアのフィルタ
- 自由度の高いテクスチャ形式
  - 4ビット/8ビットパレット形式、DXT1圧縮、他
- 完全にパースペクティブ・コレクト (色を含む)
- サブピクセルでの正確さ
- ピクセル毎のフォグ
- アルファ・ブレンド

以前の構成

- 深いパイプライン (200ステージ)
- 常に全てのステージを通らなければならない
- OpenGL型の高速テクスチャに最適化
- パイプラインは常にクロックしている
- 高速だが非常に多くの電力を消費
- 1億ピクセル/秒を処理するのに約750mW

(約200パイプステージ)

完全に新しい構成
非常に小さな電力消費

(約50パイプステージ)

- 自由度の高いフラグメントALU
- ラスタ – フラグメントの生成とループの管理
- パイプラインは活動状態でのみ使用
- 低消費電力
  - 1億ピクセル/毎秒で50mW以下
  - 実際のゲーム中
- 拡張可能な構成

なぜジオメトリ?

- 現在のハンドヘルド処理装置
  - Arm 7/Arm 9/+
  - クロックスピード: 50Mhz – 400Mhz
  - 浮動小数点無し
  - ホストのバスがDRAMのバスと共有されている
  - 限られたシステムメモリ
- なるべく多くの処理をGPUへ
  - 電力消費効率が良い
  - パフォーマンス向上

短いパイプラインでの電力節約

書き込み回数/ピクセル= 1
テクスチャ付
ブレンド無し
Z無し

書き込み回数/ピクセル= 4

1. テクスチャ付、ブレンド無し
2. テクスチャ付、ブレンド無し、Zあり
3. テクスチャ付、ブレンド無し、Zあり
4. テクスチャ付、ブレンドあり、Zあり

簡単なシーンでは全てのポリゴンに対して、フォグ、ブレンド、アルファ、Zは必要ない

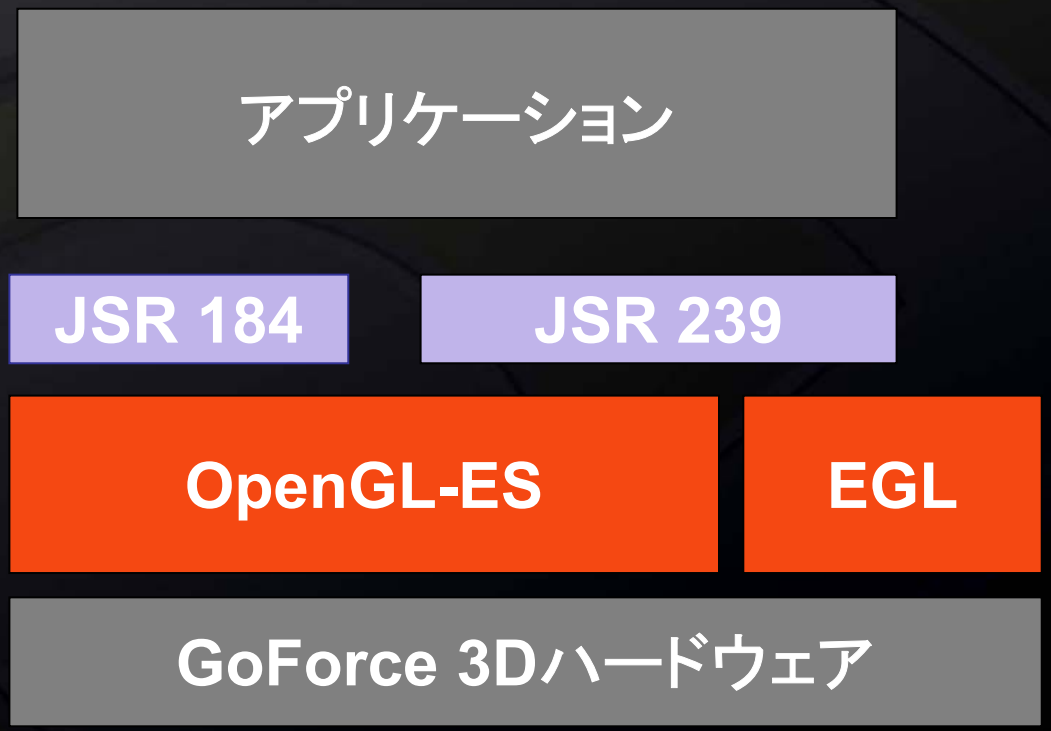
豊富な2D機能

- 一色での塗りつぶし
- ソースの複写
- アルファ・ブレンド
  - 全てのピクセルに対して固定のアルファ値
- 16x16パターンのフィル
- 線の描画
  - サブピクセルでの正確さ
- クリップと透明
  - クリップの内部と外部がサポートされている

*n*Powerテクノロジ

- 使用されていないパイプラインの自動停止
- アーキテクチャでの通常、待機、スリープのモードによる電力管理
- いくつものレベルでの、高度な電力管理
- 低電圧での動作

Javaプログラム・モデル



ネイティブなプログラム・モデル

ミドルウェアによるプログラム・モデル

OpenGL ES 1.0とOpenGL

- ほとんどOpenGL 1.3
- 無くなったもの
  - ディスプレイ・リスト
  - glBegin/glEnd
  - Texgen
  - 環境マップ
  - エバリュエータ
- 追加されたもの
  - 固定小数点タイプと関数
  - バイトタイプがより広範囲で使用可

OpenGL ES 1.1とOpenGL

- OpenGL 1.5をもとに

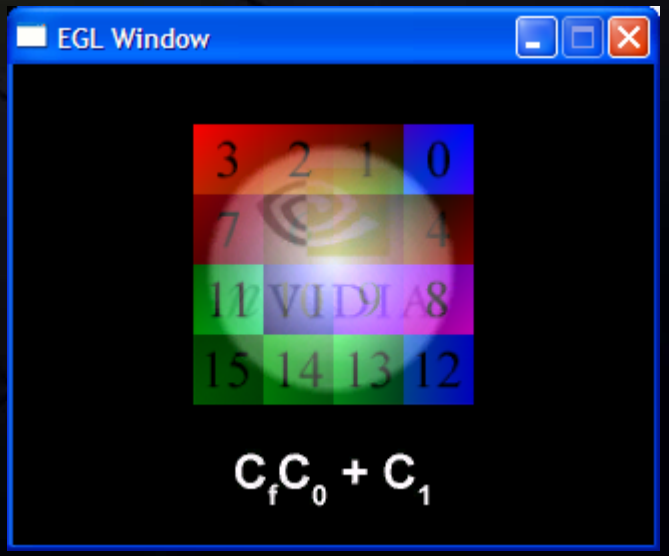
- ES 1.0に追加した機能
  - 頂点バッファ・オブジェクト
  - 自動ミップマップ生成
  - より強力なテクスチャ合成処理
  - ユーザー指定のクリップ平面
  - ポイント・スプライトとポイント・スプライト配列
  - 動的ステートの問い合わせ

Direct3Dm

- 一般公開はされていない
- Microsoftと協力している

NVIDIAハンドヘルドSDK

- デモ
  - OpenGL ES (101)
  - 機能デモ



- 移植レイヤ
  - OpenGL-ESからOpenGLへ
  - PC上で動作
  - 組み込み機器が始めての開発者が開発を始められる

NVIDIA HHDK (つづき)

- .NETデモ・ウィザード
  - x86のウィンドウズとARMリナックスで動作するアプリケーションの雛形を作成
- ツールとライブラリ
  - DXT1圧縮ツール
  - DXT1画像読み込みライブラリ
  - 固定小数点演算ライブラリ –ARMに最適化
- 文献 – GoForce 3D概要

開発キット

- もうすぐリリース!
- NVIDIAハンドヘルド開発プログラムで登録してください
  http://developer.nvidia.com
- Email
  handset-dev@nvidia.com

実装例: Bubble

- もともとはGeForce 256デスクトップGPU用に作られたデモ (circa 2000)

- 変形、反射をする球体
  - スプリングによる物理シミュレーショ
  - 環境マップ
- GoForce 3Dへの移植
  - 目標: GoForce 3Dネイティブのグラフィクス・アプリケーション開発を実証

実装例: Bubble

- デモ

Bubble: 概要

- 球体モデル – 頂点とエッジで構成
- 作用する力
  - 瞬間的な “つっつき” の力
  - “もどろうとする” 力
  - 弾力のある “エッジ” の力
  - 外側へ “膨らむ” 力
- 力の作用 – 速度、位置、法線

Bubble: 変形

- 浮動小数点でのシミュレーション – 非常に遅い
- プロファイラで問題のある領域を確認
- 固定小数点演算に切り替え(s15.16)

- 固定小数点 – 範囲と精度の兼ね合い
  - 他の形式や定数倍を使う

Bubble: 環境マップ

- もとの物はキューブマップと反射Texgenを使った
  - ES 1.0ではどちらもサポートしていない
- 双方物面マップと手動でのテクスチャ座標生成を用いた(固定小数点)

Bubble: テクスチャ・メモリの使用

- 各シーンは8つのテクスチャを使用
  - 2 – 256x256テクスチャ(ミップマップあり)
  - 6 – 256x256テクスチャ(ミップマップ無し)

R5G6B5 – テクセルごとに16ビット

= 786432 + 349524 = 1.08Mb

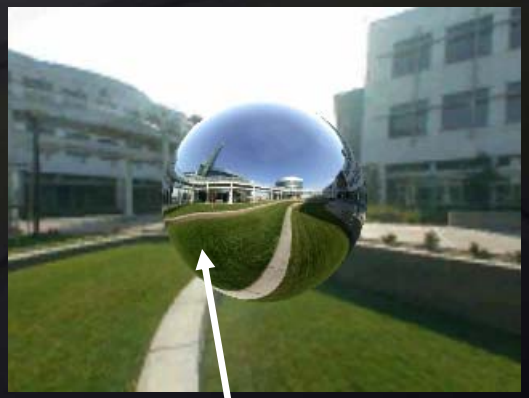
DXT1 – テクセルごとに4ビット

= 196608 + 87381 = 0.27Mb

DXT1なら高画質でR5G6B5の25%の大きさ

Bubble: 画質

バイリニア



高周波数

トライリニア

シルエットの
近くで高周波数

トライリニアと
LODの切り取り

最高の画質 *

* SGIS_texture_lodを使用

NVIDIA 3D高品質デモ

x86/ウィンドウズでOpenGL-ESラッパーを用いて動作

GoForce 3Dハードウェアの能力をエミュレート

NVIDIAデベロッパー・サイト

- NVIDIAハンドヘルド開発プログラムにご登録ください

developer.nvidia.com

質問?

Handset-Dev@nvidia.com

Bubble: How it Works

- Sphere Model – Set of Vertices and Edges
  - Vertex
    - Position
    - Normal
    - Velocity
    - Average Velocity – average “neighborhood” velocity
    - Home Position – vertex “home” resting position
  - Edge
    - Pair of vertex indices
    - Home Length – initial edge length

Bubble: How it Works

- Deformation – apply forces to update model
  - Vertex
    - Position
    - Normal
    - Velocity
    - Average Velocity
    - Home Position
  - Edge
    - Pair of vertex indices
    - Home Length

Multi-Step Process...

Bubble: Deformation

- Step 1 – Updating the Velocities
- Adjust based on spring forces
  - “Homeward” force
  - “Outward” force
  - “Edge” force (i.e. elasticity)

```
foreach vertex
    vel += HomeForce( home – pos )
    vel += OutwardForce( normal )
foreach edge
    vert[v0].vel += EdgeForce( vert[v0].pos – vert[v1].pos )
    vert[v1].vel += EdgeForce( vert[v1].pos – vert[v0].pos )
```

Bubble: Deformation

- Step 2 – Filter Velocities
  - Compute Average Velocities
  - Apply Filter – vel = 0.9 * vel + 0.1 * avg
- Step 3 – Update Positions
- Step 4 – Apply Drag to Velocities
- Step 5 – Compute Normals
  - Iterate over all triangles, use cross-product of edges

Bubble: Poking

- Requires Instantaneous velocity update
- Find closest point to “pick ray”
  - Eye Pos: (0,0,0)
  - Pick Ray: (screen_x,screen_y, -near)
- Apply inward pulse force based on distance

p.vel += PulseForce(distance(closest.pos, p.pos))

where PulseForce(d) = k_1 * Pow(d, -20)

Bubble: Deformation (revisited)

- Step 2 – Filter Velocities
  - What happens if we don't filter the velocities?

Simulation becomes unstable.