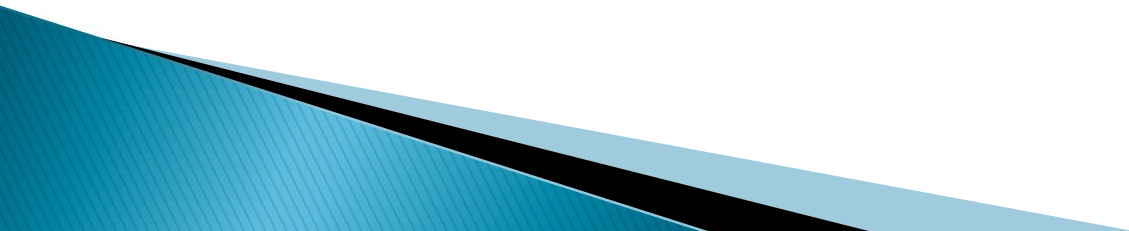


2013/08/26改訂版

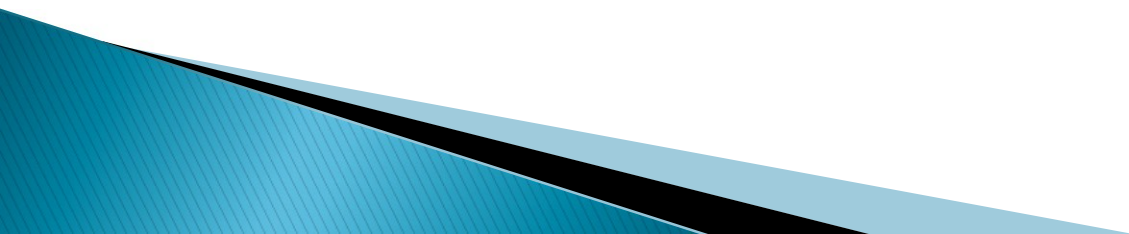
Rendering 1 h

take(@elgraiv_take)

とりあえず
レンダラを
書かねば



中身どうしよう



みんなGレンダラ書く流れらしい

➤ PPM使ってみよう

蜂須賀先生の講演会があったので
ただし実装はかなり怪しい

何か飛び道具がほしい

➤ Implicit Surface使って 髪の毛描こう

これ実はしゅうry

何をレンダリングしよう

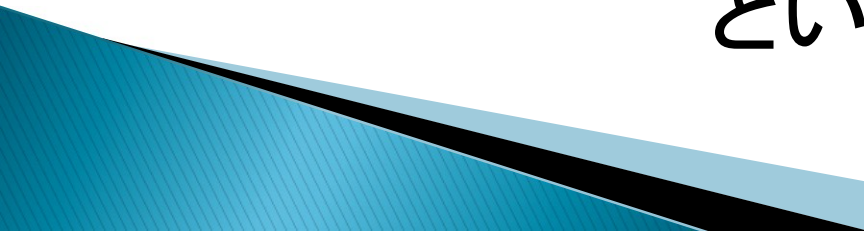
フォトンマップは光源小さいほうが有利なイメージ

→ 小さな部屋の中

髪の毛ってことは当然

→ 人間

というわけで



適当な人間と
適当な部屋を
レンダリングしよう

具体的には？

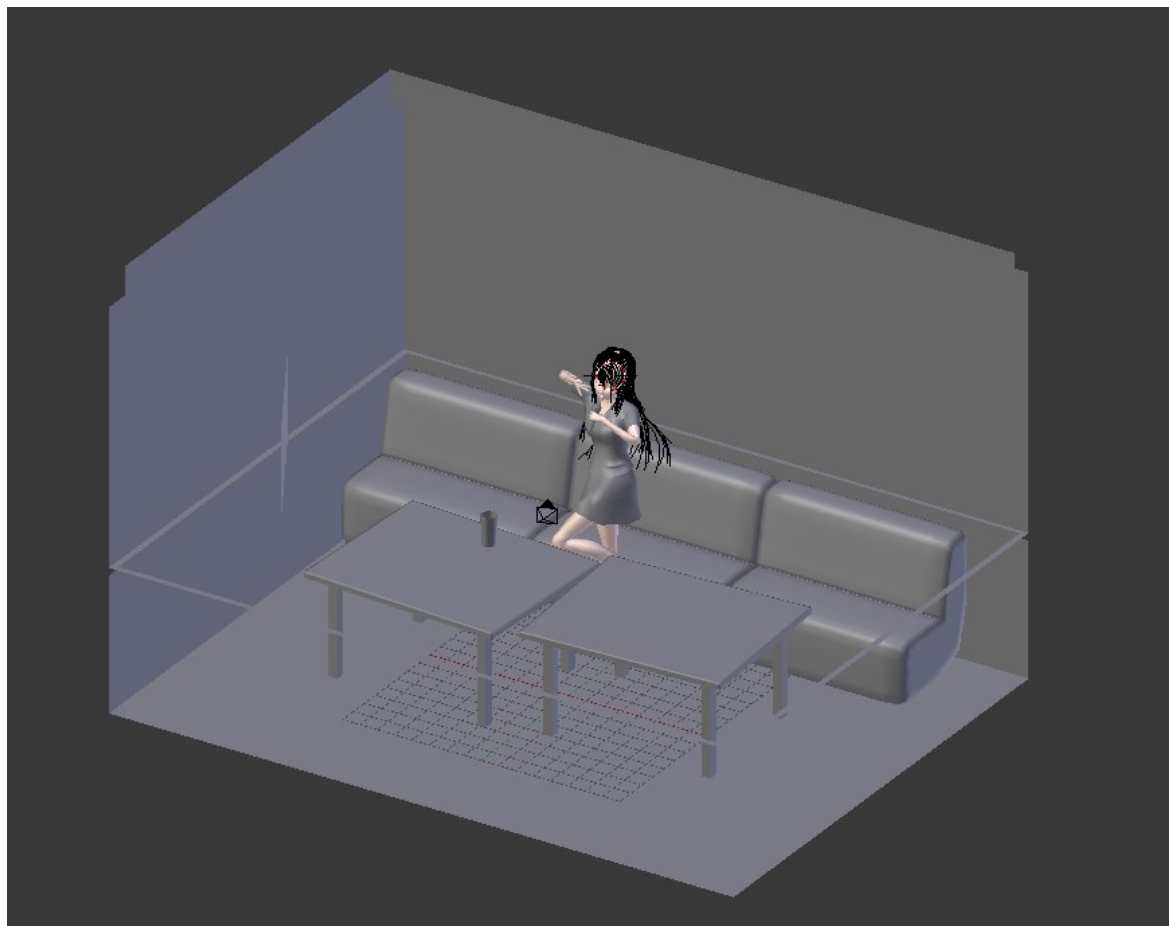
要求仕様一覧

- 髪の毛長いほうがいいから女の子
- というか男レンダリングしても面白く無いから女の子
- 髪揺れたところにしたいから多少動きがある感じ
- あまりものは多くないけど何もないわけじゃない部屋
- 光源はあまり多くないほうが嬉しい
- 窓からの光とか勘弁

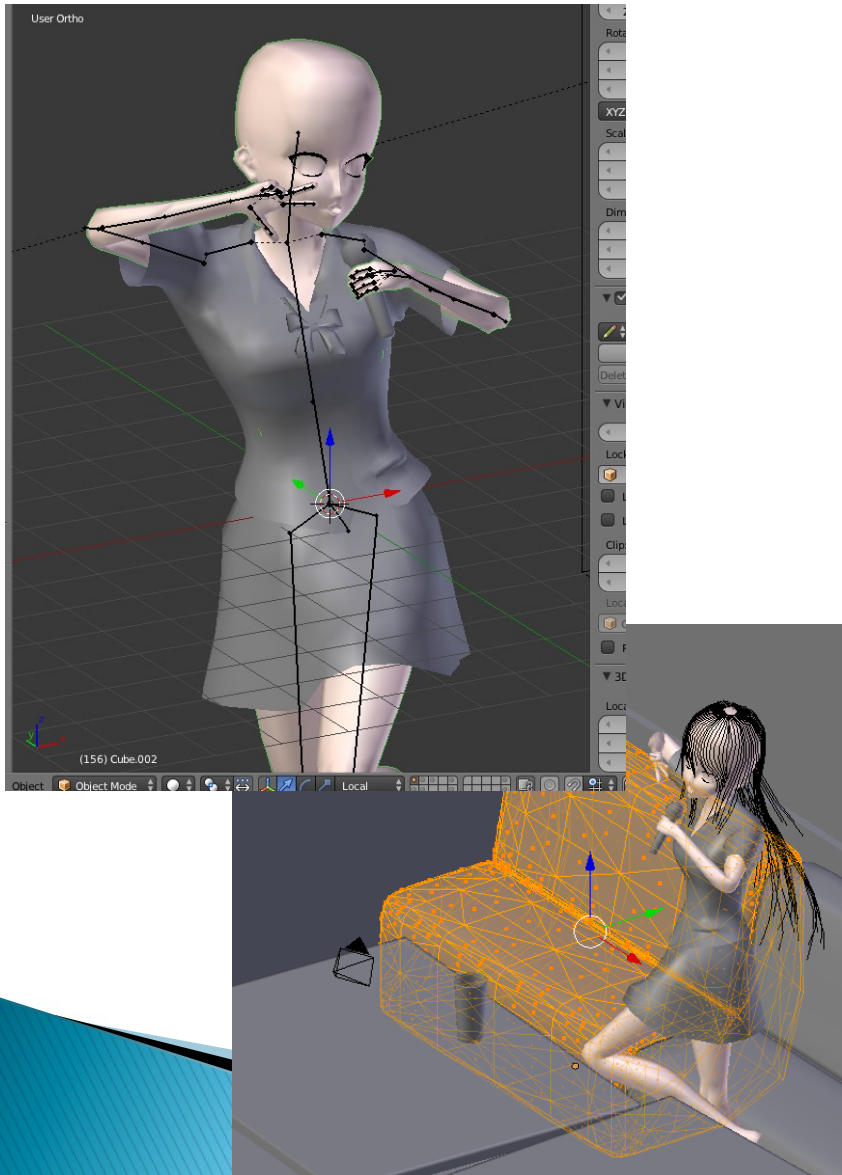
地下牢にも

カラオケで歌う女の子とかいいんじゃない？

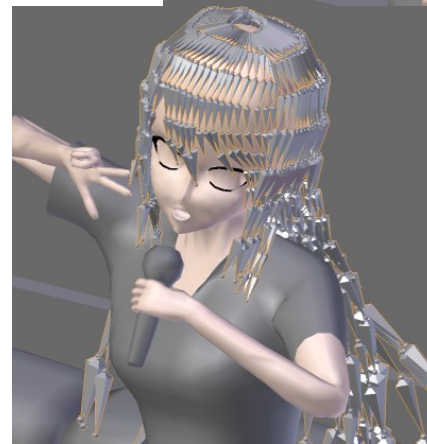
というわけでこんな感じ



シーンはみんな大好きBlenderで作成



髪の毛はスケルトンのみ



ボーンにカスタムパラメータをつけて無理やり作成

面倒なので全部オレオレ形式で出力or変換

```
11 fp.write(pack('@i', len(mdata.vertices)))
12 for v in mdata.vertices:
13     fp.write(pack('@f', v.co.x))
14     fp.write(pack('@f', v.co.y))
15     fp.write(pack('@f', v.co.z))
16     fp.write(pack('@f', v.normal.x))
17     fp.write(pack('@f', v.normal.y))
18     fp.write(pack('@f', v.normal.z))
19 fp.write(pack('@i', len(mdata.polygons)))
20 for f in mdata.polygons:
21     fp.write(pack('@i', f.vertices[0]))
22     fp.write(pack('@i', f.vertices[1]))
23     fp.write(pack('@i', f.vertices[2]))
24     if len(mdata.uv_layers) > 0:
25         uv = mdata.uv_layers[0].data
26         for li in range(f.loop_start, f.loop_start + f.loop_total):
27             fp.write(pack('@f', uvs[li].uv[0]))
28             fp.write(pack('@f', uvs[li].uv[1]))
29     else:
30         fp.write(pack('@f', 0.0))

71 break
72 pbn = bn
73 bn = pbn.children[0]
74 f.write(str(nodeNum) + "\n")
75
76 f.write(str(rootB.head_local.x) + "," + str(rootB.head_local.y) + "," + str(rootB.head_local.z) + ",")
77 f.write(str(rootB["r"]) + "," + str(rootB["v"]) + "\n")
78 cloc = rootB.head
79 bn = rootB
80 while(True):
81     log.flush()
82     if len(bn.children) < 1:
83         break
84     pbn = bn
85     bn = pbn.children[0]
86     cloc = bn.head_local
87     f.write(str(cloc.x) + "," + str(cloc.y) + "," + str(cloc.z) + ",")
88     f.write(str(bn["r"]) + "," + str(bn["v"]) + "\n")
89
```



Pythonスクリプトで適当にエクスポート
メッシュの情報は頂点座標, 法線, UVのみ
オブジェクトごとに別ファイルで出力
マテリアルはプログラム中でハードコーディング
スケルトンは必要なパラメータを書き出し

テクスチャも
ヘッダとか圧縮とかない形式

レンダリング結果はこんな感じ



コンペのものからパラメータを少し変えて
約6時間レンダリングしたもの

部屋

Implicit Surfaceの髪

女の子

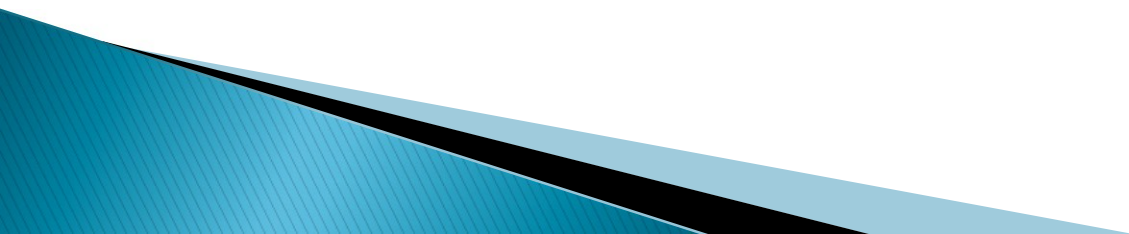


申し訳程度の「GI使ってますよ」アピール

ほんの少しだけ技術的な話

Progressive Photon Mapping(PPM)

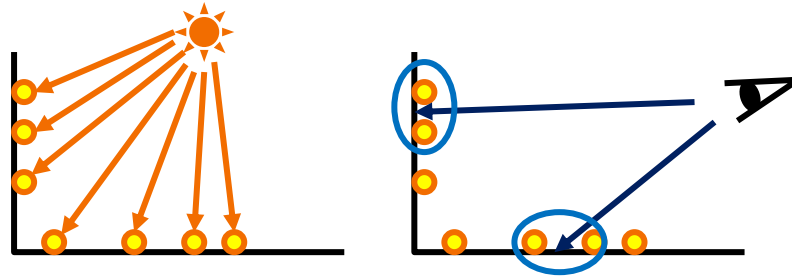
Skeleton-based Implicit Surface Hair



Progressive Photon Mapping

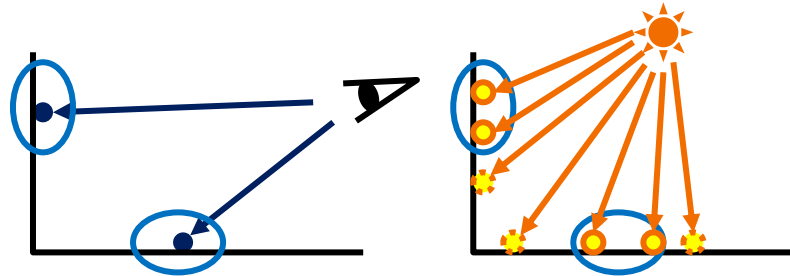
簡単に言うとレイトレーシングを第一パスにしたフォトンマッピング

フォトンマッピング



最初にフォトンをばら撒きその後レイトレで回収する

PPM



最初にレイトレで観測点を置き
その後フォトンをばら撒きながら観測点で回収する

使い終わったフォトンは捨ててしまっても構わないので
大量のフォンを飛ばしてもメモリを食わない
計算をいつでも打ち切れるので今回のルールでも使いやすい

マルチスレッド化

フォントレーシングのパスではフォトン毎に並列

しかし

スレッド毎に完全独立で計算することができない
(観測点のフォトン回収半径の更新などがあるため)

フォトン回収部分は排他制御が必要
でも一回一回全部やってたら並列化の効果が出ない



スレッドごとに回収するフォトンと観測点の組でバッファを作って
ある程度たまってから一気に適用することに

それでもシーンやパラメータによってはCPU使用率が100%にならない.....

何かいい方法やそもそも間違いがあったら教えて下さい

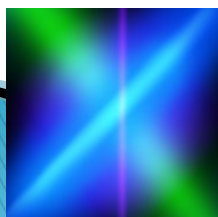
Skeleton-based Implicit Surface Hair

スケルトンからアニメキャラのような立体的な髪を作る手法



正直なところアニメーションさせないと有り難みがあまりない

各ノードをスプラインで補間したスケルトンと
メタボールライクな濃度関数を使ったImplicit Surfaceを
レイマーチングで直接レンダリング
後は適当に描いた画像をBRDFとして使ってシェーディング



今回使ったBRDFテーブル

BRDFテーブルについて

本当はFDTD法を使ってシミュレーションした何かの構造色を使おうと思ったけど
レギュレーション違反らしいのと使ったところでネタ以上の意味がないのと
そもそも結果がそんなに綺麗になるわけじゃないためオミット

その他

適当実装のBVH

OpenMPでのお手軽並列

拡張性がよくなってるといいなあみたいなクラス構造

途中画像の出力と1時間で打ち切るための監視スレッド

などなど



終わり

