Blender 実行環境構築とデモ

AnyTech 岩井

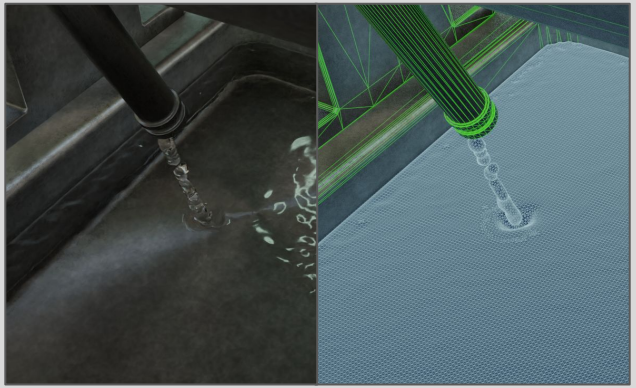
なぜCGを学習データに用いるのか?

現実の映像データを用いることの難所

1. 学習に適したデータが少ない
2. 手ブレ、不適切なライティング、カメラアングル
3. 速度ベクトル、粘度の真値がわからない
4. 流体部分のアノテーション作成が困難

CGデータを用いることのメリット

1. 撮影環境を自由に設定可能
2. 速度ベクトル、粘度は流体シミュレーションから取得可能
3. 流体部分のアノテーションが自動で完了



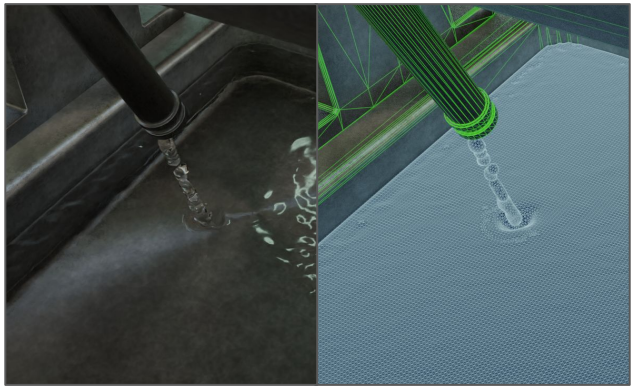
なぜCGを学習データに用いるのか?

現実の映像データを用いることの難所

1. 学習に適したデータが少ない
2. 手ブレ、不適切なライティング、カメラアングル
3. 速度ベクトル、粘度の真値がわからない
4. 流体部分のアノテーション作成が困難

CGデータを用いることのメリット

1. 撮影環境を自由に設定可能
2. 速度ベクトル、粘度は流体シミュレーションから取得可能
3. 流体部分のアノテーションが自動で完了



使用するCG制作ソフト

Blenderとは

1. 無償の3DCG作成ソフト
2. 写実的な映像制作が可能
3. セットアップの簡便さ

https://www.blender.org/

Blenderのおすすめポイント

- 物理シミュレーション
- Ubuntuでも動作
- 一部機能はCUIでも実行可能
- Python APIが充実
- 物体のセグメンテーション、速度ベクトル、深度が取得可能

使用するCG制作ソフト

Blenderとは

1. 無償の3DCG作成ソフト
2. 写実的な映像制作が可能
3. セットアップの簡便さ

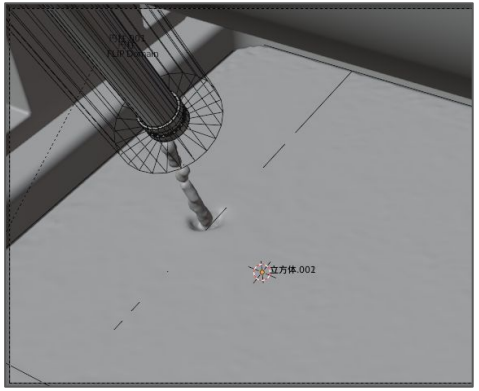
https://www.blender.org/

Blenderのおすすめポイント

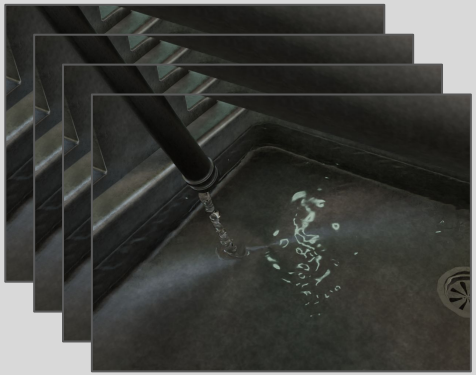
- 物理シミュレーション
- Ubuntuでも動作
- 一部機能はCUIでも実行可能
- Python APIが充実
- 物体のセグメンテーション、速度ベクトル、深度が取得可能

BlenderのCG作成フロー

1. **ローカルでの作成作業工程**
- CGモデリング
- 流体シミュレーションのパラメータ調整

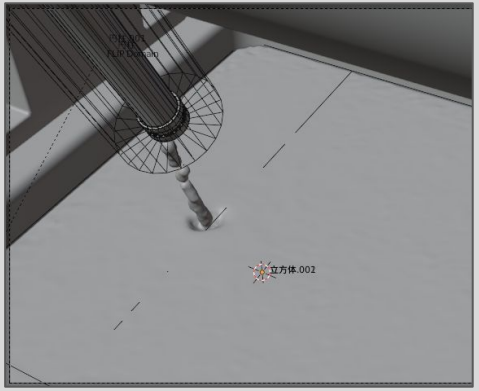


2. **PC任せの工程（ABCI活用）**
- 1フレームずつレンダリング
- OpenCV等で動画化

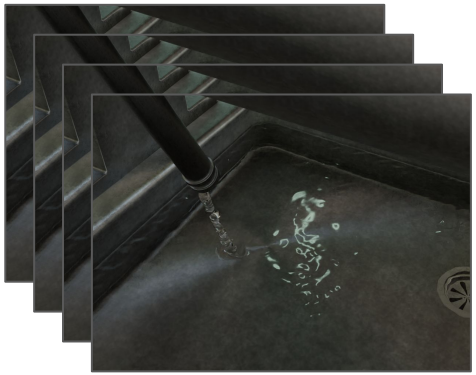


BlenderのCG作成フロー

1. **ローカルでの作成作業工程**
   - CGモデリング
   - 流体シミュレーションのパラメータ調整

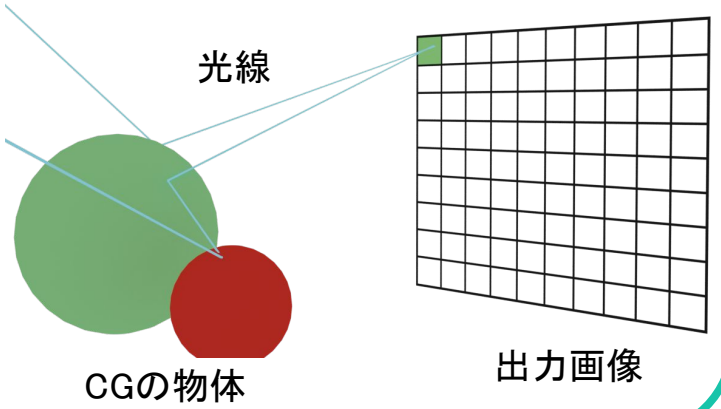


2. **PC任せの工程（ABCI活用）**
   - 1フレームずつレンダリング
   - OpenCV等で動画化



レンダリングの仕組みと問題点

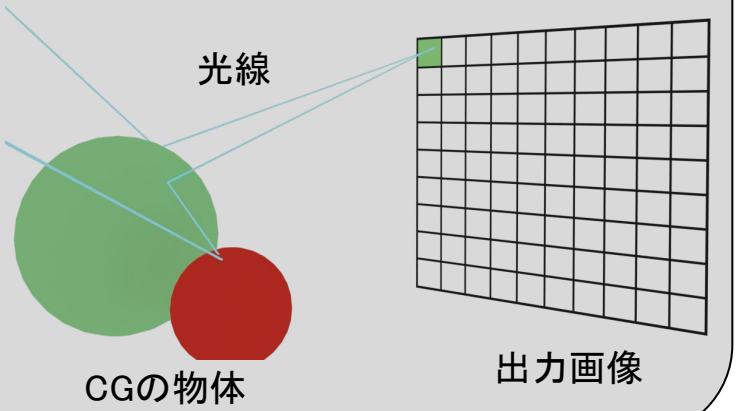
・レイトレーシングによるレンダリング
- 各ピクセルから複数の光線を飛ばして
  光の経路を計算し各ピクセルの色を決定する
- 写実的な映像が作れる
- GPUによる並列計算が可能



・レイトレーシングの問題点
- 綺麗にするにはより多くの光線のサンプル(512や1024)が必要
- 画像1枚に10~90秒かかる
  30fps, 1分の動画だと5~45時間
- 常にGPU使用率100%

レンダリングの仕組みと問題点

・レイトレーシングによるレンダリング
- 各ピクセルから複数の光線を飛ばして
  光の経路を計算し各ピクセルの色を決定する
- 写実的な映像が作れる
- GPUによる並列計算が可能



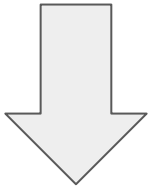
・レイトレーシングの問題点
- 綺麗にするにはより多くの光線のサンプル(512や1024)が必要
- 画像1枚に10~90秒かかる
  30fps, 1分の動画だと5~45時間
- 常にGPU使用率100%

オンプレとABCIの比較

	オンプレGPUサーバ	ABCI
マルチタスク	レンダリング以外のタスク実行ができない	空き状況次第でマルチタスクが可能 
時間あたりの動画数	長時間かけて1つ	並行して複数動画のレンダリングが可能 
GPU数	1枚	4枚 
ランニングコスト	電気代が高い	ポイント消費
レンダリング時間	12時間超えたくない	最大72時間 

ABCIでのレンダリング準備・実行概要

1.Blender実行環境の構築
- UbuntuベースのDocker imageを作成
- Docker imageをsingularityに変換



2. レンダリング実行
- blender -b <***.blend> -P render.py

ABCI上でのBlender実行環境構築

```
FROM nvidia/cuda:11.4.0-runtime-ubuntu20.04



RUN wget
https://mirror.freedif.org/blender/release/Blender3.1/blender-3.1.2-linux-x64.tar.xz
RUN tar Jxfv blender-3.1.2-linux-x64.tar.xz  && \
   mv blender-3.1.2-linux-x64  /opt/blender









ENV NVIDIA_VISIBLE_DEVICES  all
ENV NVIDIA_DRIVER_CAPABILITIES  compute,utility,graphics
```

1.GPUが使えるImageをベースに作成

2.Blender 3.0 or laterを
ダウンロード＆インストール

3.レンダリングにOptixを使う設定
これがないとオンプレ1GPUよりマルチGPUの方が
大幅に遅くなる

レンダリング用サンプルコード

render.py

```
import bpy

bpy.context.scene.cycles.device = 'GPU'
prefs = bpy.context.preferences
prefs.addons['cycles'].preferences.get_devices()
cprefs = prefs.addons['cycles'].preferences
cprefs.compute_device_type = 'OPTIX'


for device in cprefs.devices:
    device.use = False
    if device.type == cprefs.compute_device_type:
        device.use = True


bpy.ops.render.render(animation=True)
```

1.BlenderのPythonライブラリ

2.レンダリングにOptixを使う設定

3.レンダリングにCPUを使わない設定

4.レンダリング開始

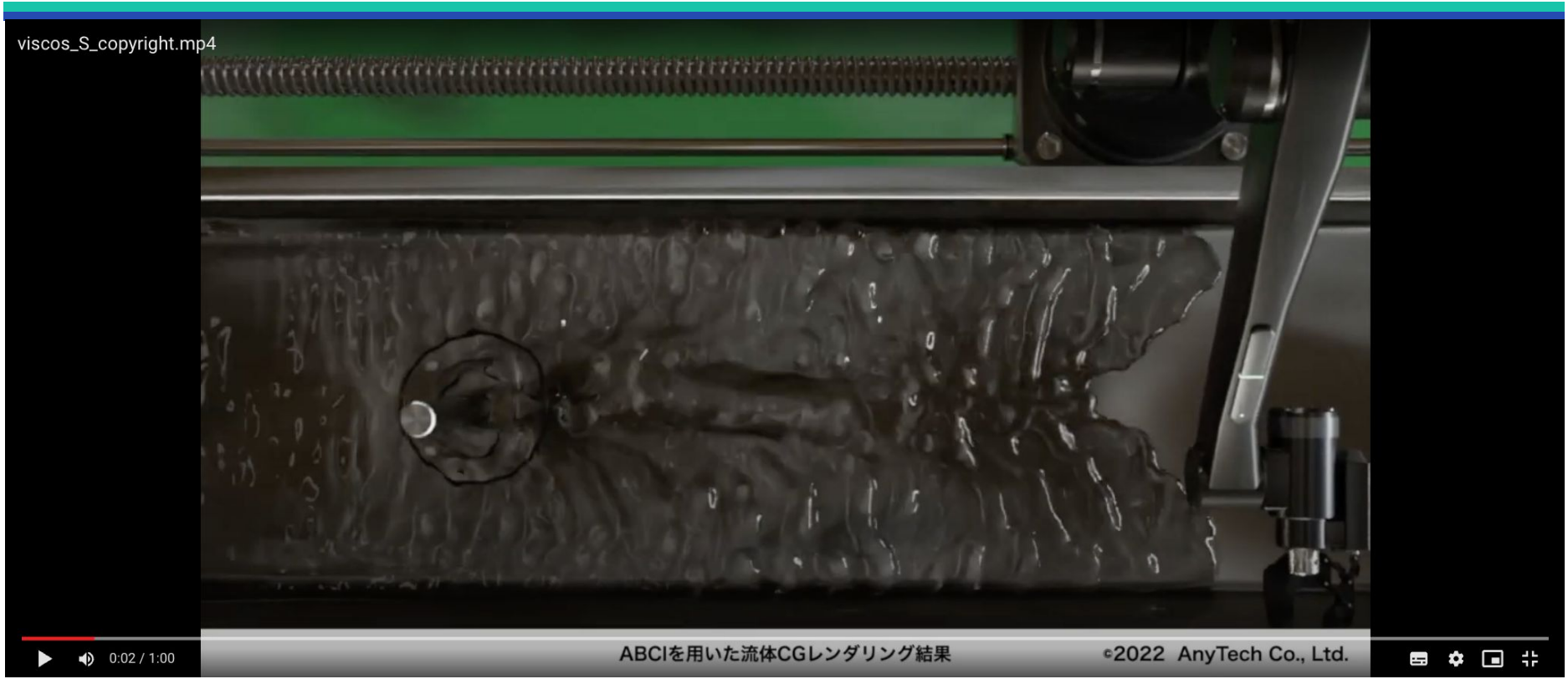
解像度と使用ポイント数

	rt_F（4GPU, 1pt/時）	rt_G.small（1GPU, 0.3pt/時）
1280 x 720, 128サンプル	8.5 秒/枚 1分のCGで5pt	**11 秒/枚** **1分のCGで1.8pt**
1280 x 720, 512サンプル	5.45 秒/枚 1分のCGで3pt	26.2 秒/枚 1分のCGで4.2pt
3840 x 2160, 512サンプル	**29.7 秒/枚** **1分のCGで15pt**	108 秒/枚 1分のCGで16.2pt

【状況に応じて最適な計算資源タイプを選択】
・時間重視ならrt_F
・ポイント重視なら場合によりrt_G.small

制作動画紹介

制作動画デモ1



※ウェビナー中のみ再生可

制作動画デモ2

※ウェビナー中のみ再生可

ご視聴ありがとうございました