POINT CLOUD RENDERER

PccAppRenderer v6.0

Apr. 2021

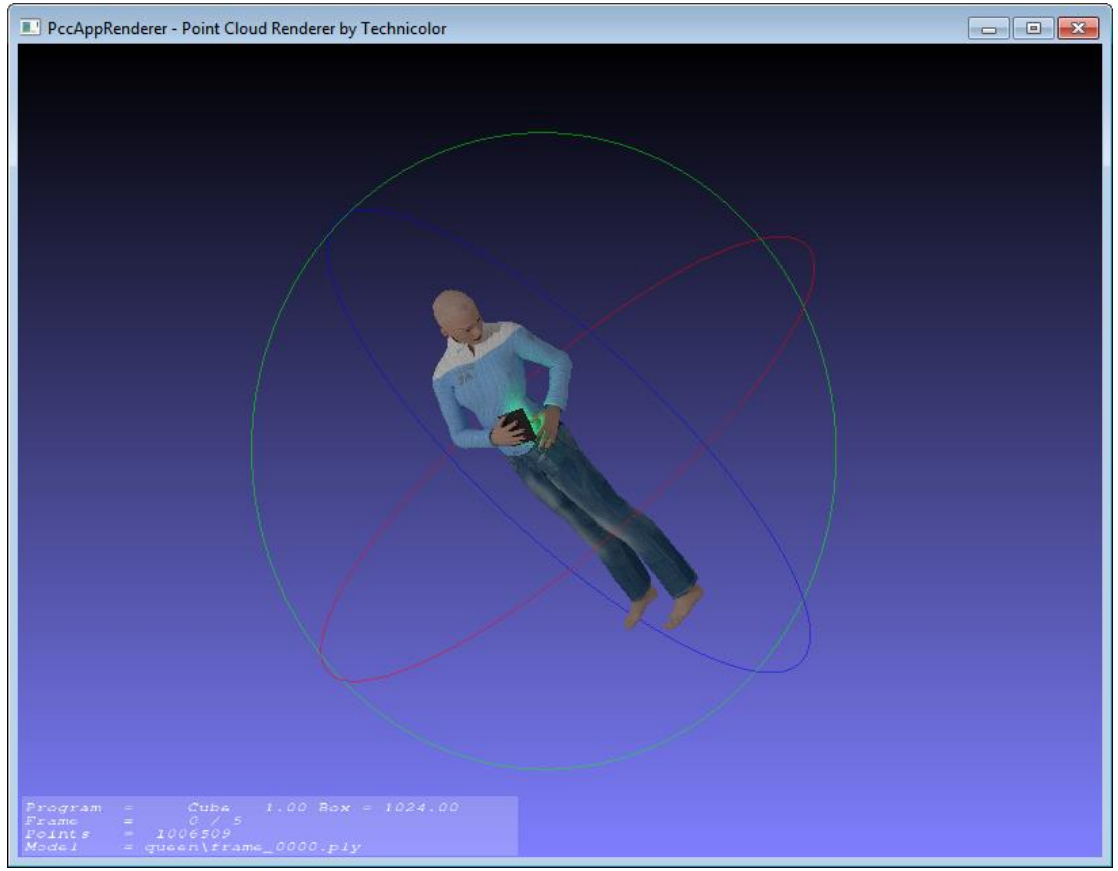
INTERDIGITAL®

Main Functionalities

Render static and dynamic point clouds

Key features

- Basic renderer functionalities
  - Navigation model/video
  - FPS control
  - …
- Camera Path
  - Use a predefined path
  - Register your own path
  - Replay your path with your model
- Output RAW File
  - Register the rendering in a video RGB file
- Synchronization between renderers
- Scriptable



Input format

- Based on PLY input format
- Format element
  - ascii and binary_little_endian are supported
- Vertex element
  - Other flags than [x y z] [nx ny nz] [r g b a] are skipped
  - [x y z] are mandatory

Ply files are read in alphabetical order

Application : Inline Help

```
PccAppRenderer configuration: input parameters must be:
  -v,   --version=0       Print version.
  -h,   --help=0          Print help.
        --config=...      Parse configuration file
  -f,   --PlyFile=""      Ply input filename
  -d,   --PlyDir=""       Ply input directory
        --SrcFile=""      Source Ply filename (used for comparison)
        --SrcDir=""       Source Ply directory (used for comparison)
  -b,   --binary=0        Create temp binary files
  -o,   --RgbFile=""      Output RGB 8bits filename (specify prefix file name)
  -x,   --camera=""       Camera path filename
  -y,   --viewpoint=""    Viewpoint filename
        --spline=0        Interpolate the camera path by splines
  -n,   --frameNumber=1   Frame number
  -i,   --frameIndex=0    Frame index
        --fps=30          Frames per second
  -a,   --align=0         Align (0:X, 1:-X, 2:Y, 3:-Y 4:Z, 5:-Z)
        --width=800       Window width
        --height=600      Window height
        --posx=-1         Window position X
        --posy=-1         Window position Y
        --size=1          Point size
        --type=2          Point type:
                            Point cloud: 0: cube,
                                         1: circle,
                                         2: point
                            Mesh:        0: wireframe,
                                         1: surface+wireframe
                                         2: surface
                                         3: point
```

Application : Inline Help

```
...
      --monitor=0       Monitor to display the window
      --background=0    Window background index
      --depthMap=0      Display depth map
-p,   --play=0          Play the sequences
      --playBackward=0  Play sequence forward and backward.
-r,   --rotate=0        Auto-rotate (speed in [0;4])
      --overlay=1       Display overlay
      --synchronize=0   Synchronize multi-windows
      --box=1024        Bounding box size
      --dropdups=2      Drop same coordinate points (0:No, 1:drop, 2:average)
-c,   --center=0        Center the object in the bounding box.
-s,   --scale=0         Scale mode:    0: disable,
                                       1: scale according to the object bounding box.
                                       2: scale according to the sequence
                        bounding box.
      --floor=0         Adds grey floor under objects.
      --scenePath=""    3D background scene path (obj object).
      --sceneScale=1    3D background scene scale.
      --scenePosX=0     3D background scene position X.
      --scenePosY=0     3D background scene position Y.
      --scenePosZ=0     3D background scene position Z.
      --sceneRotX=0     3D background scene rotation X.
      --sceneRotY=0     3D background scene rotation Y.
      --sceneRotZ=0     3D background scene rotation Z.
```

Application : Launch PccAppRenderer

Static model

```
PccAppRenderer.exe -f model.ply
```

Dynamic model

Read from a directory	`PccAppRenderer.exe -d your_PLY_directroy -n 0`
Read n first frame of a model	`PccAppRenderer.exe -f model_%04d.ply -n 200`
Read n frame starting from an index of a model	`PccAppRenderer.exe -f model_%04d.ply -n 200 -i 20`
Play the model in loop	`PccAppRenderer.exe -d your_PLY_directroy -p`
Control the FPS	`PccAppRenderer.exe -d your_PLY_directroy --fps=25`

Binary mode

```
PccAppRenderer.exe -f model.ply -b
```

➔ accelerate the launching of the renderer

Application : Additional Options

Adapt Window Size (Width & Height)	`--width=1280 -height=720 | -w 0`
Choose your monitor identifier	`--monitor=1`
Choose your background	`--background=3`
Define the bounding box size	`--box=0`
Align your object	`--align=2`
Scale your object	
No scaling	`-s 0`
According to the object Bounding Box	`-s 1`
According to the sequence Bounding Box	`-s 2`
Choose if the model is center or not	`-c`
Load a camera path	`-x camera.txt`
Output in RGB files	`-o video`
Define size & type of points for the rendering process	`--size=0.05 --type=0`
Remove duplicate points	`--dropdups=2`

Command line examples

- Load one ply file.

```
PccAppRenderer -f Egyptian_mask.ply
```

- Load one ply file and scale+center the points in default viewing bounding box [0;1024]^3.

```
PccAppRenderer -f Egyptian_mask.ply -s -c
```

- Load one ply file and center the point in object bounding box [0;Xmax-Xmin][0;Ymax-Ymin][0;Zmax-Zmin].

```
PccAppRenderer -f Egyptian_mask.ply --box=0 -c
```

- Load one ply file and define the size and the type of the points for the rendering process.

```
PccAppRenderer -f Egyptian_mask.ply -s -c --size=0.05 --type=0
```

- Load N ply files from on directory.

```
PccAppRenderer -d longdress/Ply/ -n 250
```

- Load N ply files from on directory in binary mode.

```
PccAppRenderer -d longdress/Ply/ -n 250 -b
```

- Create video.

```
PccAppRenderer -d longdress/Ply/ -n  50 -b --overlay=1 -x plane_short.txt -o dec
               -p --type=2 --size=1 --overlay=0
```

Read output video examples

- Vooya

```
vooya save_20170519_15h04m33s_800x600_8bits_rgb24i.rgb
```

- FFPLAY

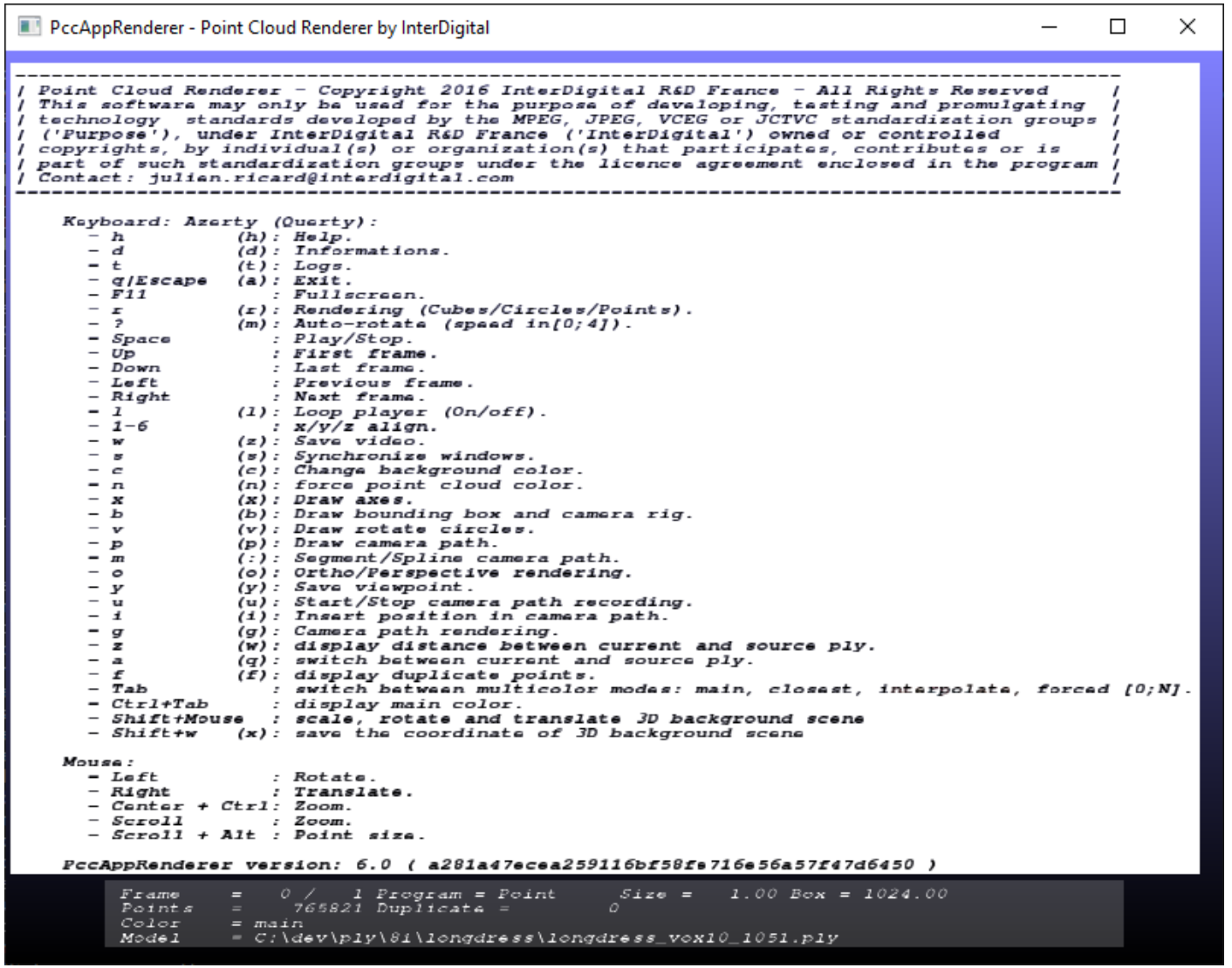
```
ffplay -f rawvideo -pix_fmt rgb24 -s 800x600
   -i save_20170519_15h04m33s_800x600_8bits_rgb24i.rgb
```

- FFMPEG

```
ffmpeg -f rawvideo -pix_fmt rgb24 -s 800x600
   -i save_20170519_15h04m33s_800x600_8bits_rgb24i.rgb
   -vcodec mpeg4 output.mp4
```

Embedded Features : Keyboard & Mouse Shortcuts

"h" : display online help in log windows



Basic Embedded Features

Zoom, Translate, Rotate, Auto-rotate

Change background color

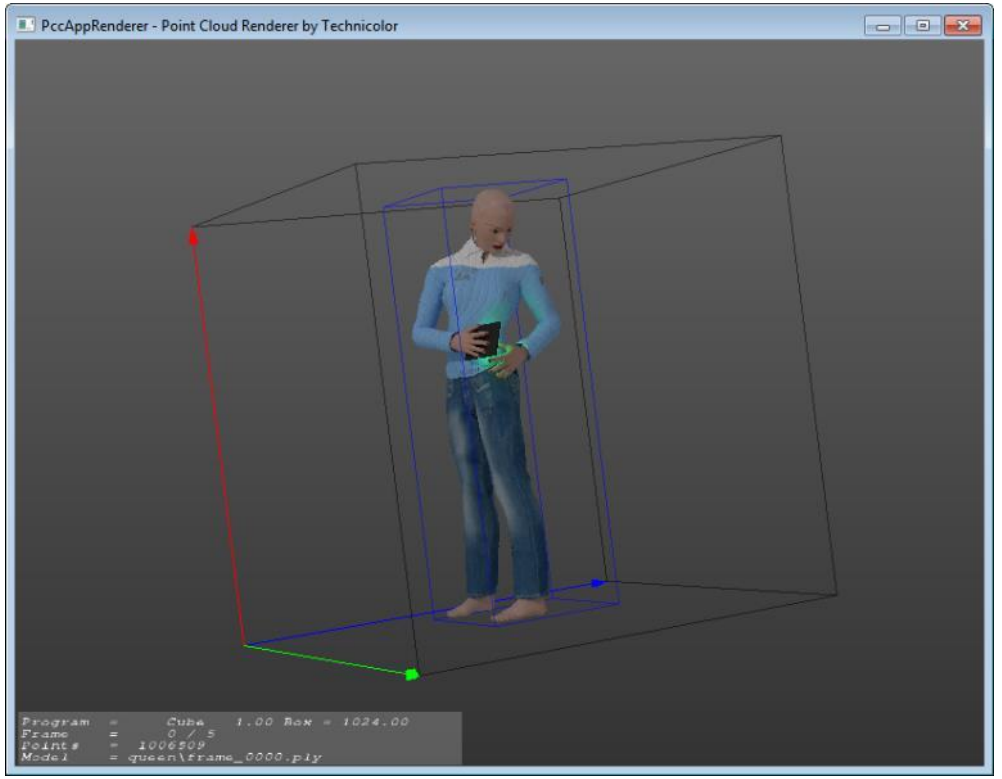
Show axes and bounding boxes

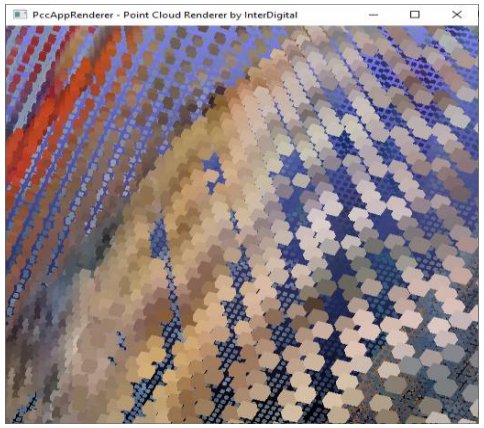
Orthographic and Perspective views

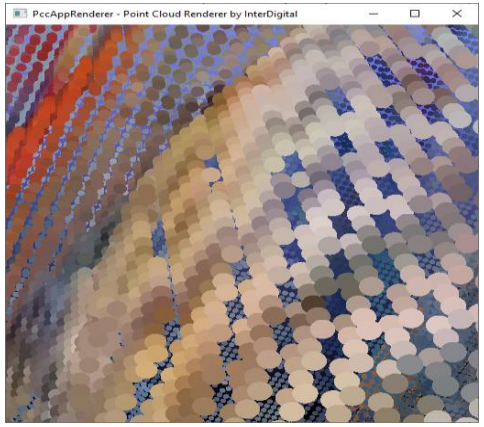
Cubes, Splats, Points with editable size

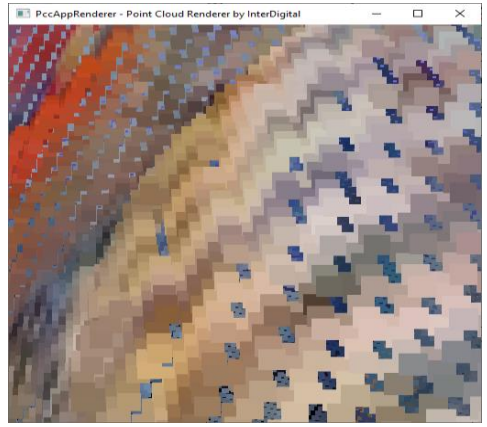
Play/Pause, Loop, Video navigation

Scriptable









Enhanced Embedded Features

Synchronization between renderer

- “a” : Synchronize windows

Camera Path

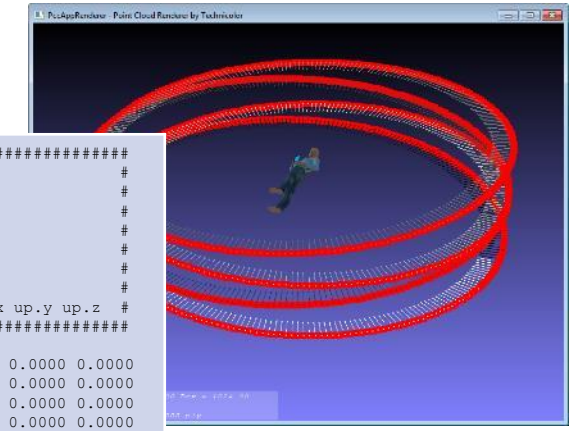
Load a camera path file

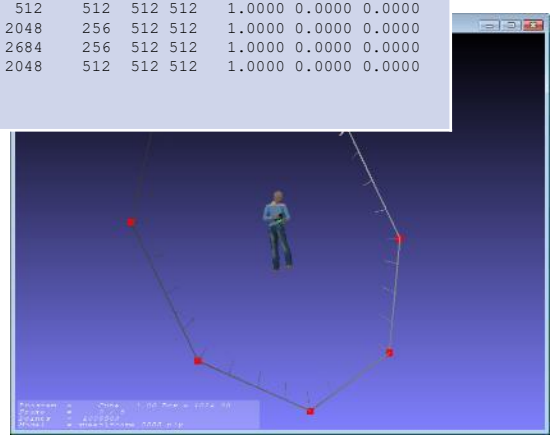
Register your own Camera Path file

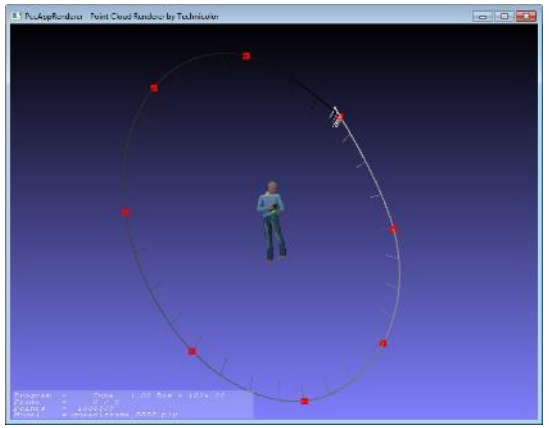
- “u” : Start/Stop the recording of the path
- “i” : Insert points into the record
- “p” : Show the current camera path
- “m”: Segment line/Spline mode
- “g” : Generate the rendering based on your path

```
#################################################################
#  Camera path file:                                            #
#                                                               #
#    Index: index of the current view                           #
#    pos  : position of the camera                              #
#    view : direction of the camera                             #
#    up   : axis of the camera                                  #
#                                                               #
# Index pos.x pos.y pos.z view.x view.y view.z up.x up.y up.z  #
#################################################################

     0  512   2048   2048    512  512 512   1.0000 0.0000 0.0000
    32  512   2684    512    512  512 512   1.0000 0.0000 0.0000
    64  512   2048  -1024    768  512 512   1.0000 0.0000 0.0000
    96  512    512  -1660    768  512 512   1.0000 0.0000 0.0000
   128  512  -1024  -1024    512  512 512   1.0000 0.0000 0.0000
   160  512  -1660    512    512  512 512   1.0000 0.0000 0.0000
   196  512  -1024   2048    256  512 512   1.0000 0.0000 0.0000
   228  512    512   2684    256  512 512   1.0000 0.0000 0.0000
   255  512   2048   2048    512  512 512   1.0000 0.0000 0.0000
```







Output video RGB file

Available in playing mode

```
PccAppRenderer.exe -d your_PLY_directory –o OutputFile
```

Available with a camera path rendering mode

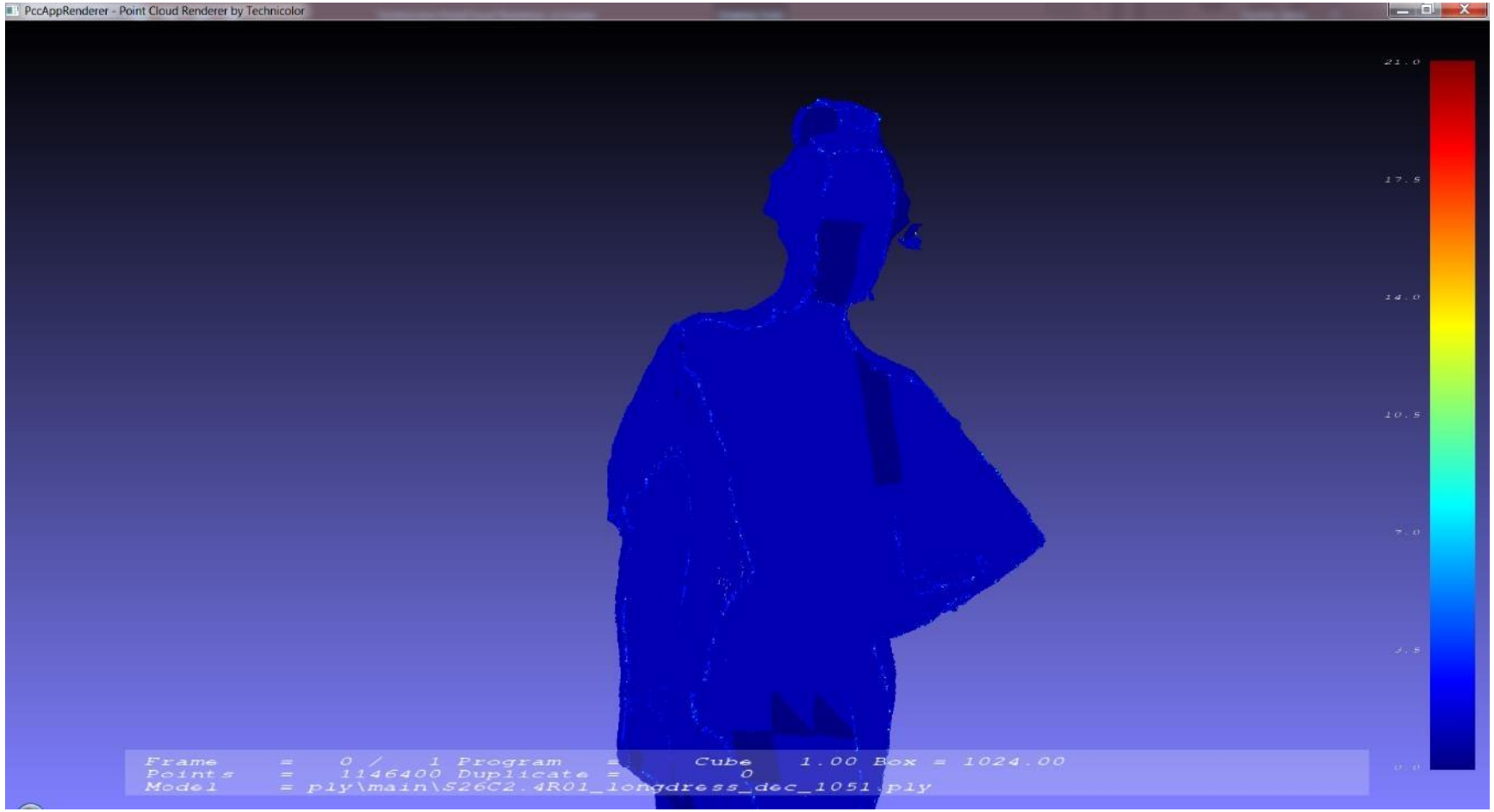
```
PccAppRenderer.exe -d your_PLY_directory –x cam.cfg –o OutputFile
```

- “w” : Record your video file on the fly

Display duplicate points

- Remove duplicate points : –dropdups=0|1|2
- Display duplicate points (‘f’ key)

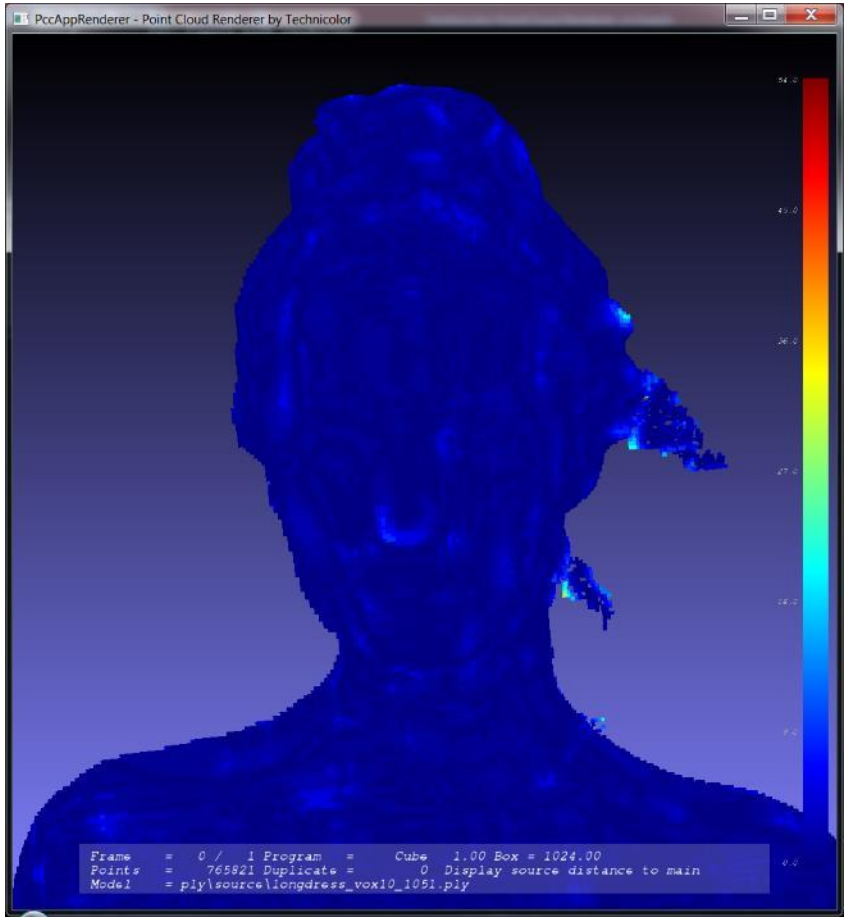
```
PccAppRenderer.exe –f file.ply --dropdups=0
```

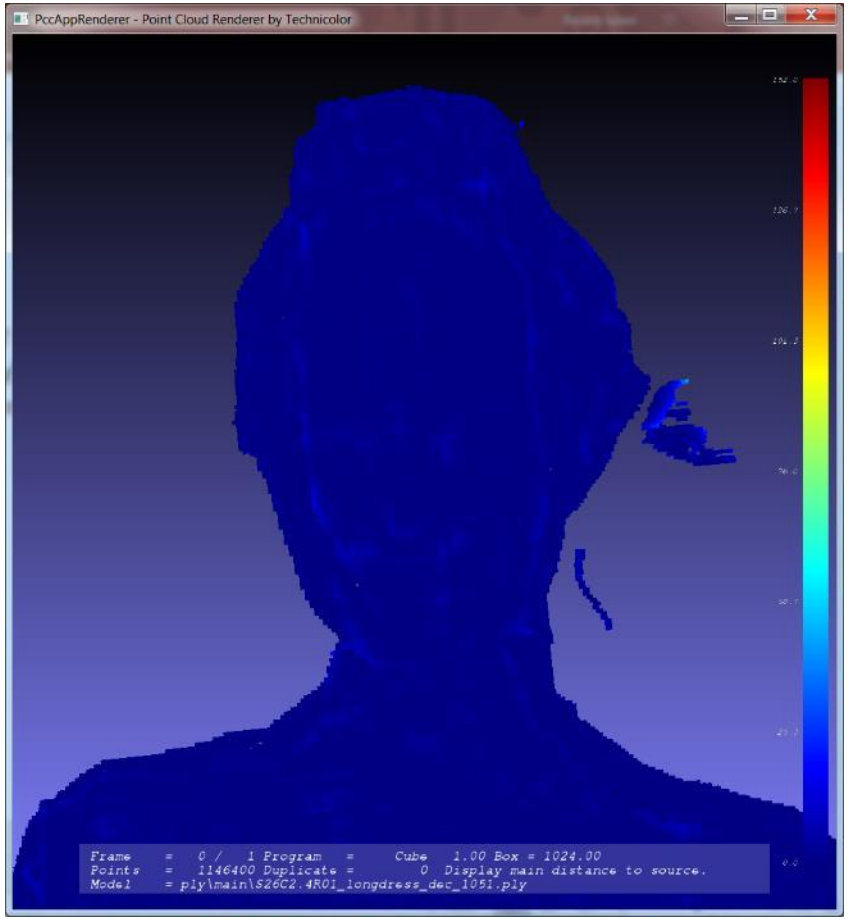


Display distance to source

- Display distance from current ply to source ply (‘z’ key)
- Switch between source and current ply (‘a’ key)

```
PccAppRenderer.exe –f file.ply –-srcFile=soure.ply
```

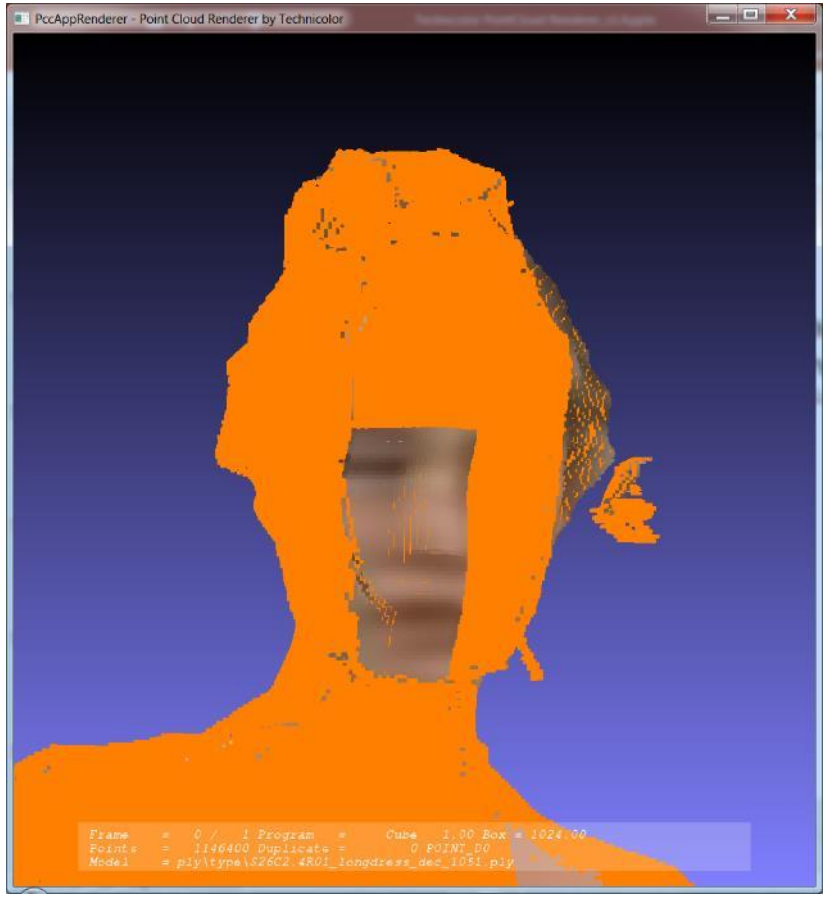


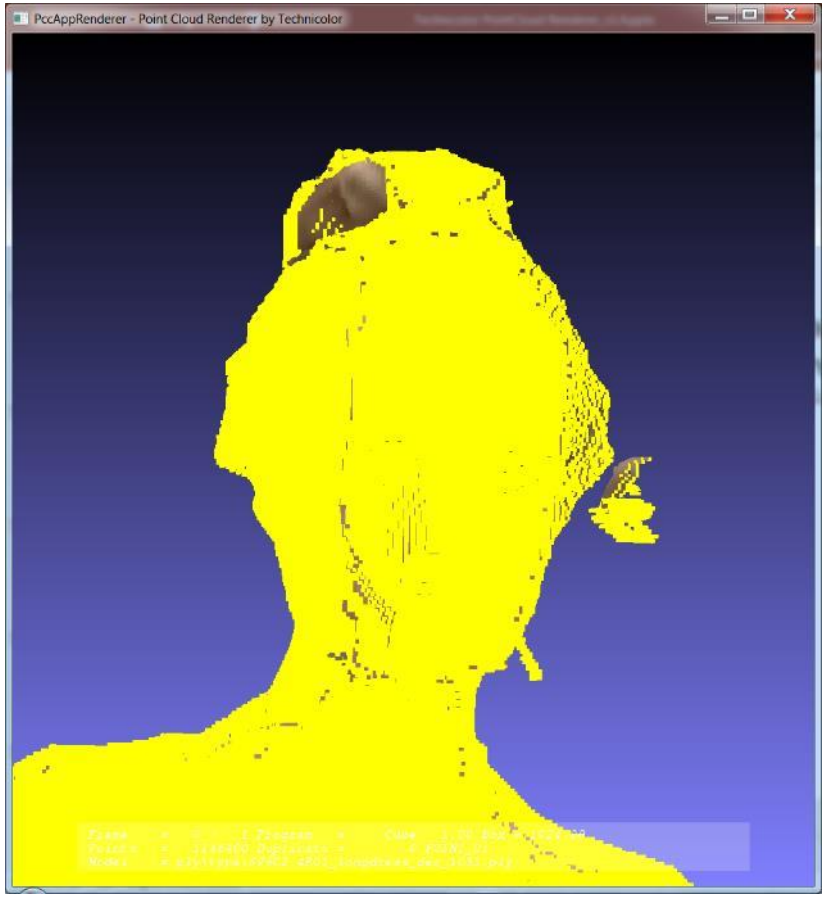


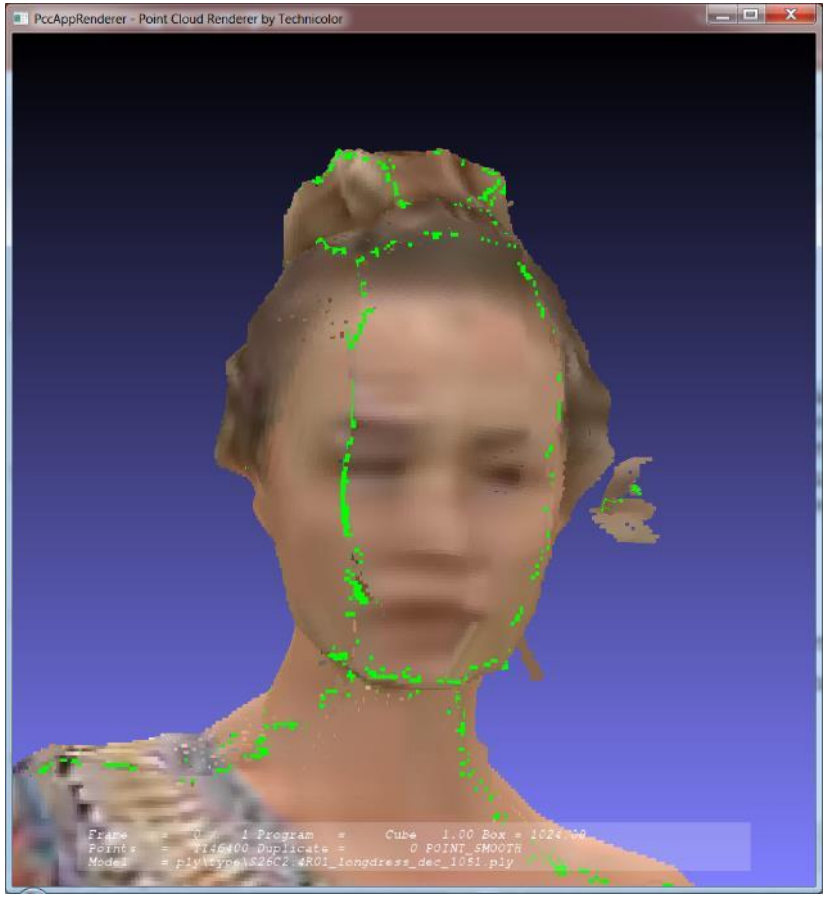
Display type of the points (TMC2 1.2)

- Display and switch between the type of the points (‘e’ key)

```
PccAppRenderer.exe -f file.ply  --dropdups=0
```

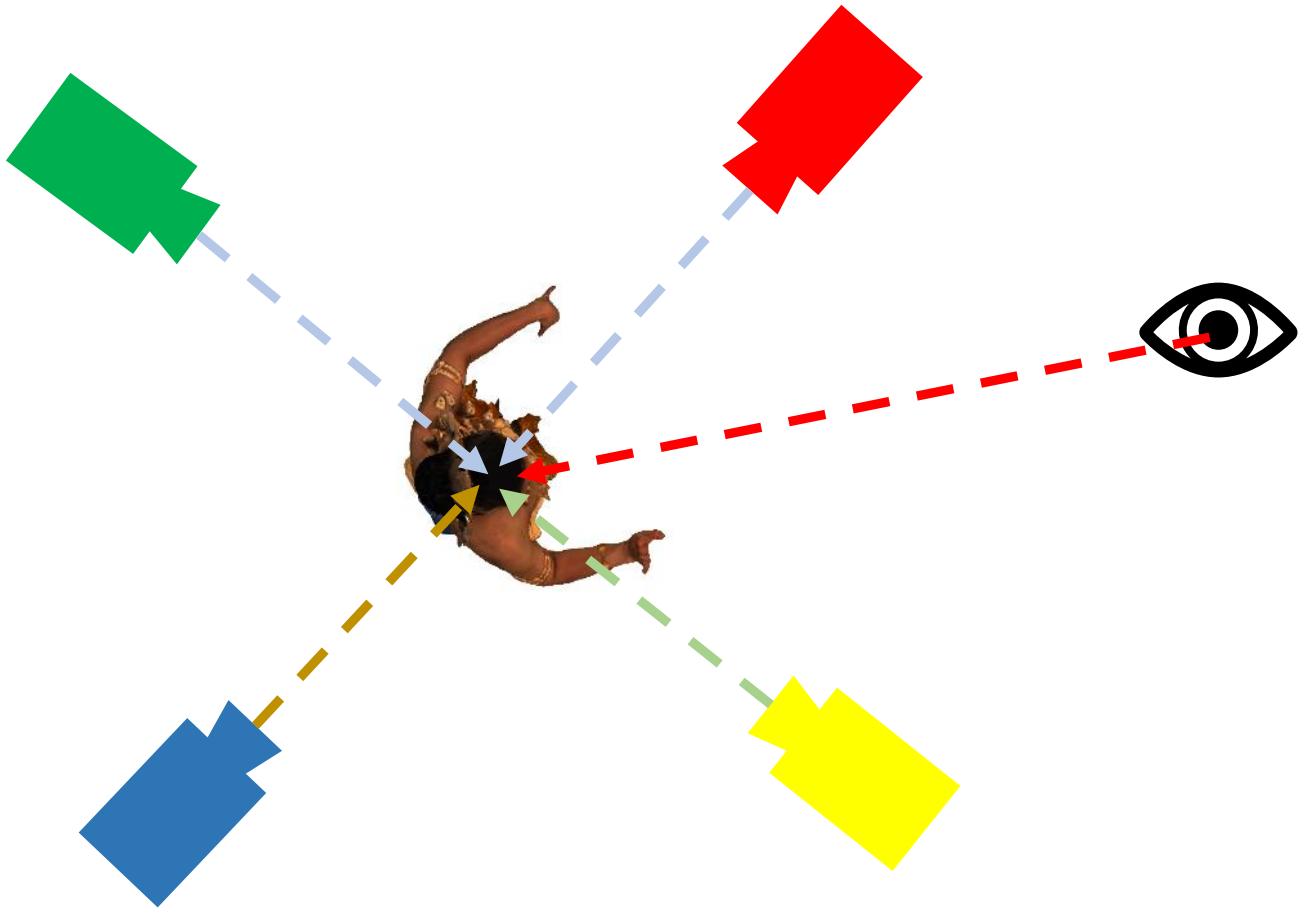






Rendering modes (1/2)

- Default
  - Main color
- Force color
  - Manually camera selection
- Closest camera
  - For each camera (i)
    - Dot product between
      - $\overrightarrow{Camera_i\ position - model\ center}$
      - $\overrightarrow{Viewer\ position - model\ center}$
    - Color of the closest camera



Rendering modes (2/2)

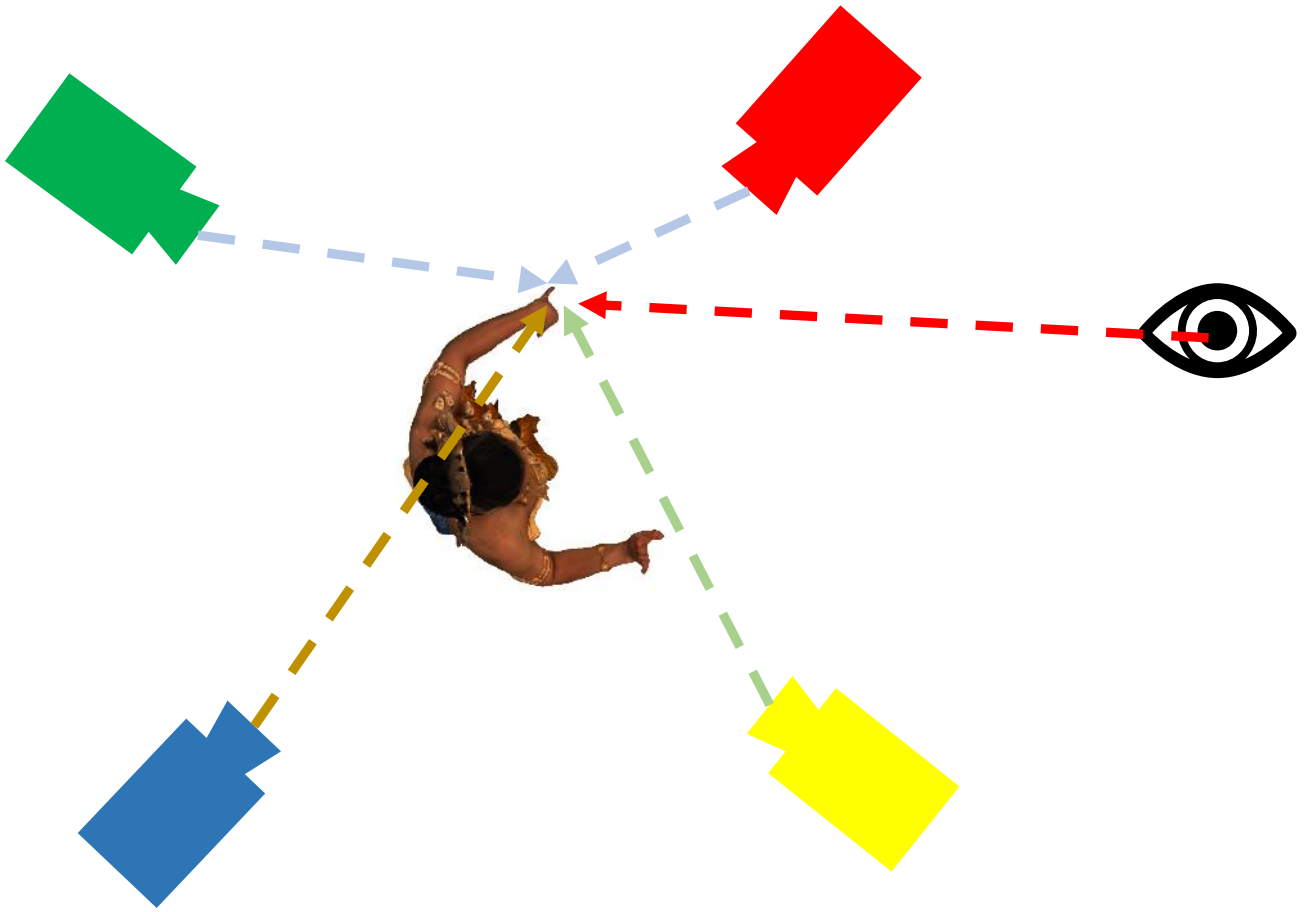
- Interpolate color by points
  - For each point (j)
    - For each camera (i)
      - Dot product (w_i) between
        - $\overrightarrow{Camera_i\ position - point_j}$
        - $\overrightarrow{Viewer\ position - point_j}$
      - Scale weight

        $\boldsymbol{w_i = max(\ 0, {w_i}^n)}$
    - Weighted sum

      $$\boldsymbol{Color_j = \frac{\sum_i w_i\ .\ Color_i}{\sum_i w_i}}$$

Note:

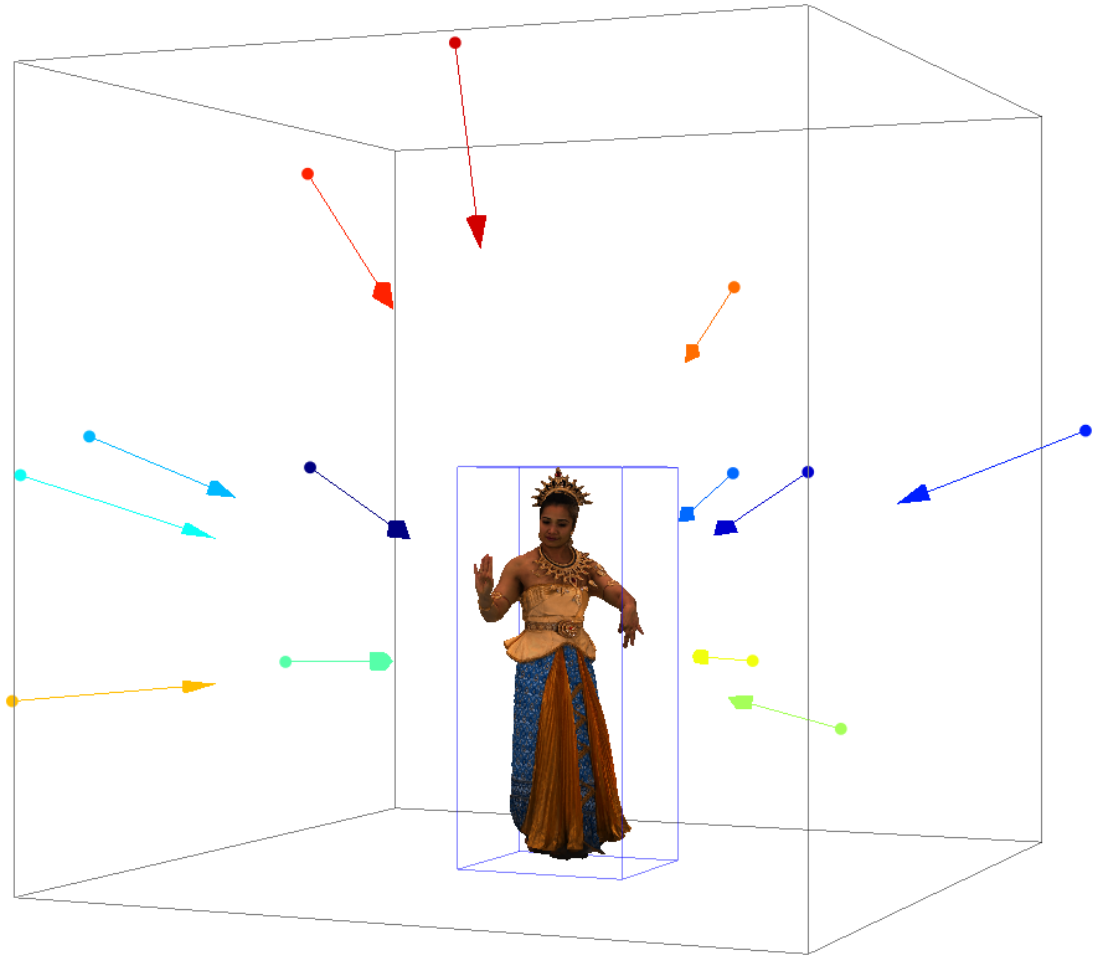
- Currently, color interpolation uses n=10



Keyboard Shortcuts

New commands:

- ‘B’: display camera rig
- ‘Tab’: switch between mod
  - Main
  - Closest
  - Interpolate
  - Camera 0
  - Camera 1
  - Camera 2
  - …
- ‘Ctrl + Tab’: back to main r

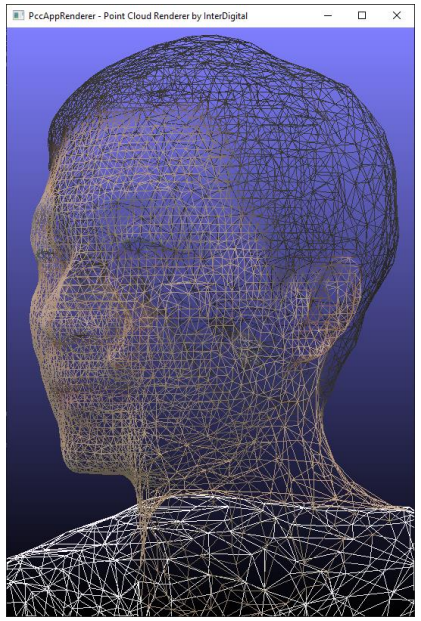


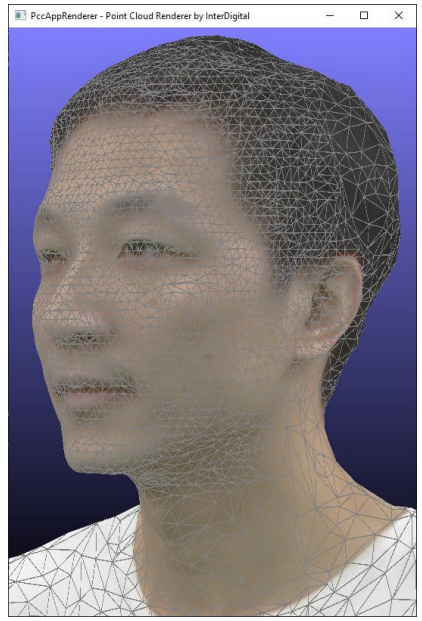
Mesh objects

- Load 3D mesh object (ply and obj):

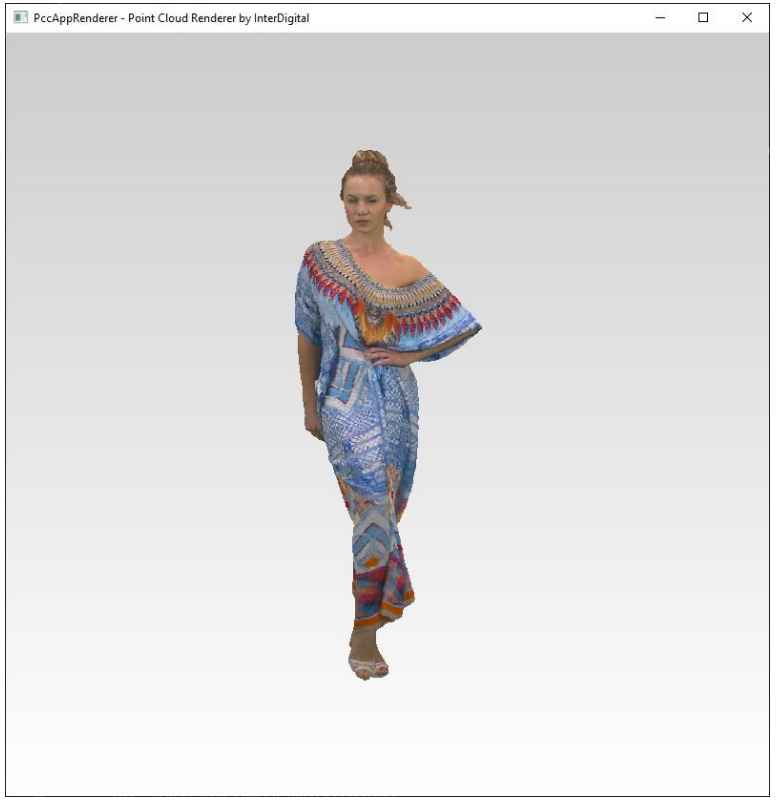
```
./PccAppRenderer -f ./basketball_player/basketball_player_%08d.obj -n 32 -i 1 -s 2 -c 1
./PccAppRenderer -d /c/dev/obj/basketball_player/ -n 32 -i 1 -s 2 -c 1
```

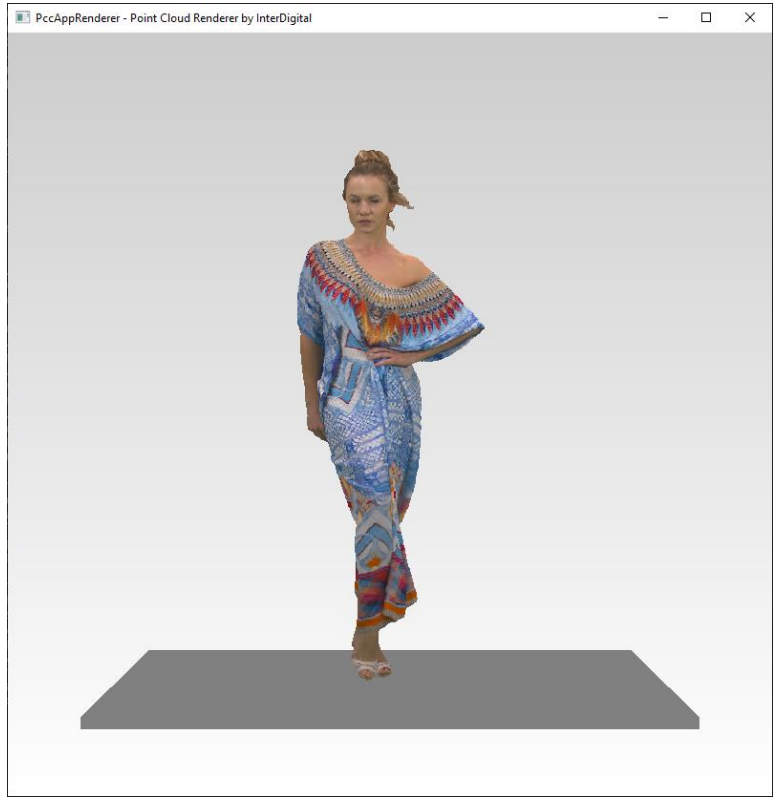
- 3 rendering shaders: surface and wireframe.

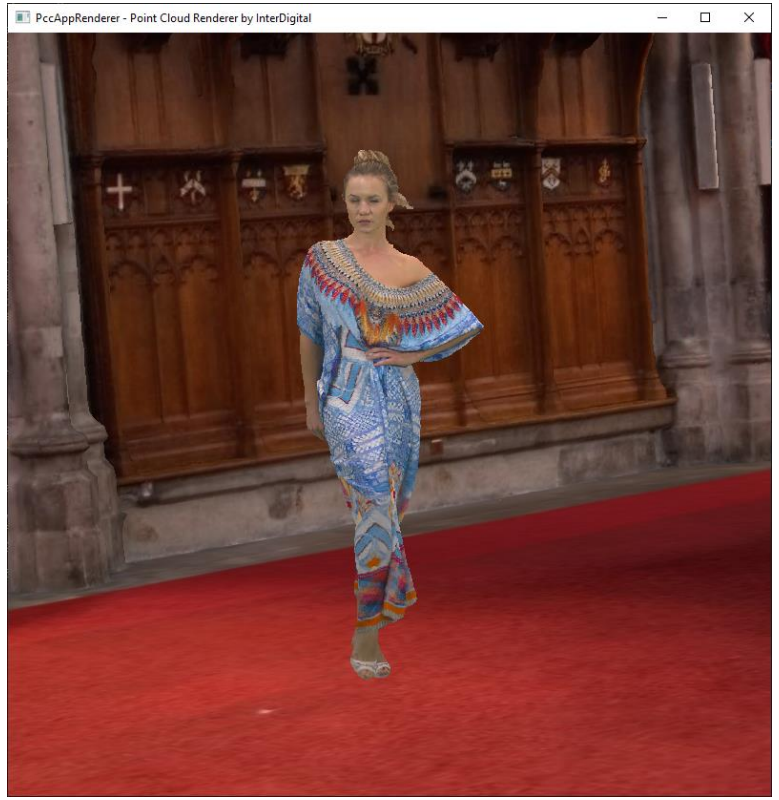




Integration of the model in a VR environment







- Example of rendering:
  - Uniform background *(left)*
  - Virtual floor *(center)*
  - 3D background scene *(right)*

Load 3D background scene

- Load 3D background scene in PccAppRenderer
  - `--scenePath=""` 3D background scene path (obj object).
  - `--sceneScale=1` 3D background scene scale.
  - `--scenePosX=0` 3D background scene position X.
  - `--scenePosY=0` 3D background scene position Y.
  - `--scenePosZ=0` 3D background scene position Z.
  - `--sceneRotX=0` 3D background scene rotation X.
  - `--sceneRotY=0` 3D background scene rotation Y.
  - `--sceneRotZ=0` 3D background scene rotation Z.

```
$ ./PccAppRenderer -d /c/dev/ply/8i/longdress/ --scenePath=model.obj
```

- Keyboard and mouse commands:
  - Shift+Mouse: scale, rotate and translate 3D background scene
  - Shift+w : save the coordinate of 3D background scene

Create video

- The script:
  - ./scripts/renderer.sh and
  - ./scripts/convert_video.sh

  could be used to create lossless video of rendering

- Example:

```
./scripts/renderer.sh  \
   -i /c/dev/ply/8i/longdress/  \
      -r 0 --width=1280 --height=720 --param=--spline=1 \
      --param=--type=0 \
      --param=--config=/d/obj/scene/scene_GuildhallMain05.txt \
      --param=--camera=camerapath_20210415-15h43m36s_0019_points.txt
```

Technical Description

- The following dependencies are used:
  - GLFW https://github.com/glfw/glfw
  - nanoflann https://github.com/jlblancoc/nanoflann
  - Assimp https://github.com/assimp/assimp
  - Stb_image https://github.com/nothings/stb/blob/master/stb_image.h
  - TBB https://github.com/oneapi-src/
  - program-options-lite

The required sub-modules are cloned from CMakeFile command

Standalone application (libraries are linked directly in it)
Compatible Windows, Linux and MacOS system
Cmake based

Compiling instructions

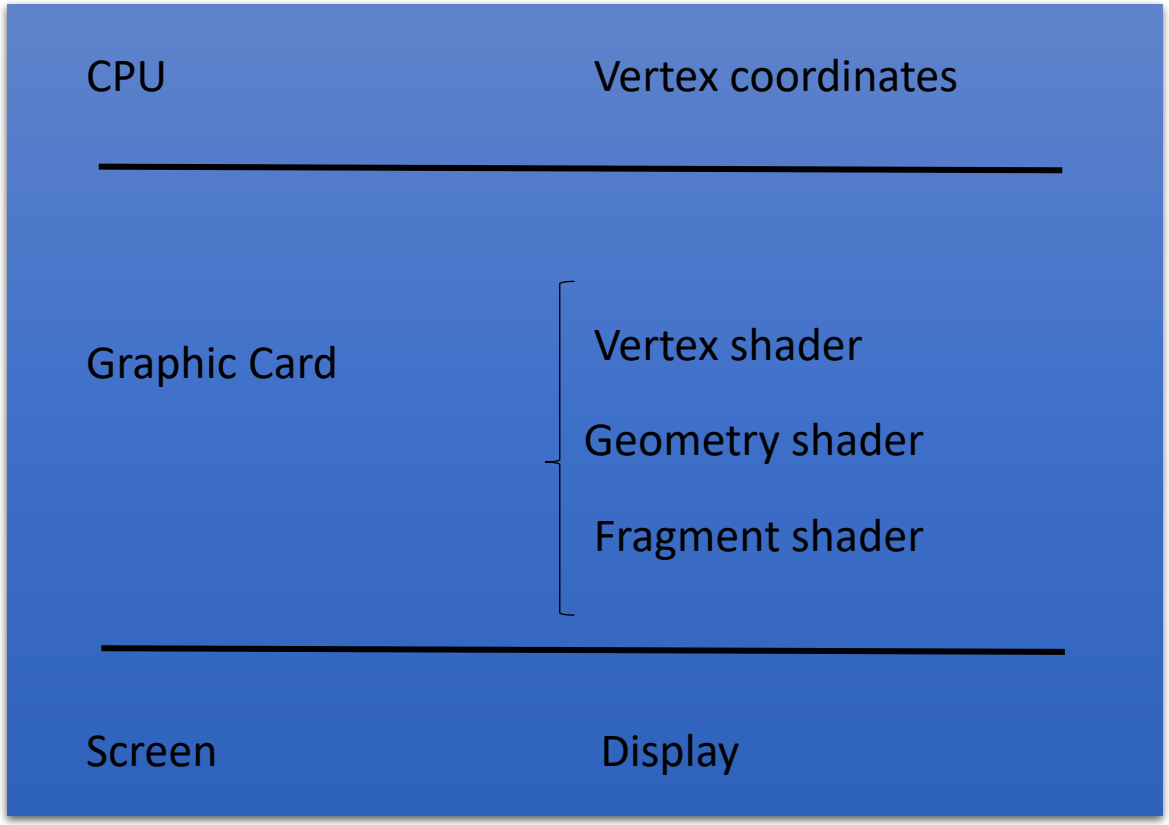
- Compilation can be performed as follows:
  - to build use the command `build.sh`.
  - to clean all object, use the command `clear.sh`
  - Or `clear.sh all` to remove all dependencies

- Don't hesitate to report any issues by the GitLab Issue tracker:

http://mpegx.int-evry.fr/software/MPEG/PCC/mpeg-pcc-renderer/-/issues

Point cloud shaders

Vertex Buffer Object organization



Point shaders
- g_pPointVertexShader
- g_pPointFragmentShader

Cube shaders
- g_pCubeVertexShader
- g_pCubeGeometryShader
- g_pCubeFragmentShader

Splat shaders
- g_pSplatVertexShader
- g_pSplatGeometryShader
- g_pSplatFragmentShader