Kinect Effect

A technical presentation by **Tristan Charpentier**

Outline

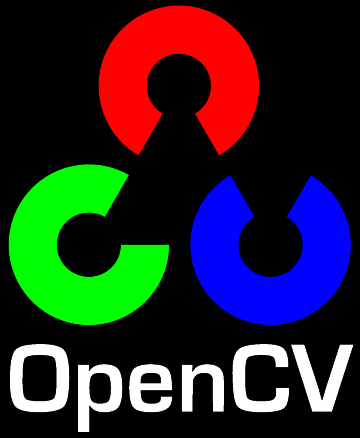
- Preview
- Dependencies
- Pipeline - highlighting 3 distinct portions
- Ray-tracing
- GStreamer
- Demo

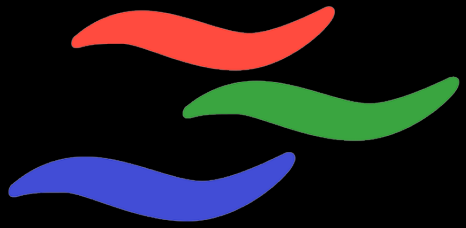
Preview

Overview

- I/O
  - Raw save, raw read, RGB save, H.264/H.265 save, livestream
- Controls
  - Web interface, XBOX controller, iOS accelerometer, MIDI controllers (DJ controller, keyboard, drums)
- 3D ray tracing renderer
  - Import and interact with 3D objects as if captured by Kinect
- Presets
  - Saving and loading presets in real time

Dependencies





thestk / **RtMidi**

TClap

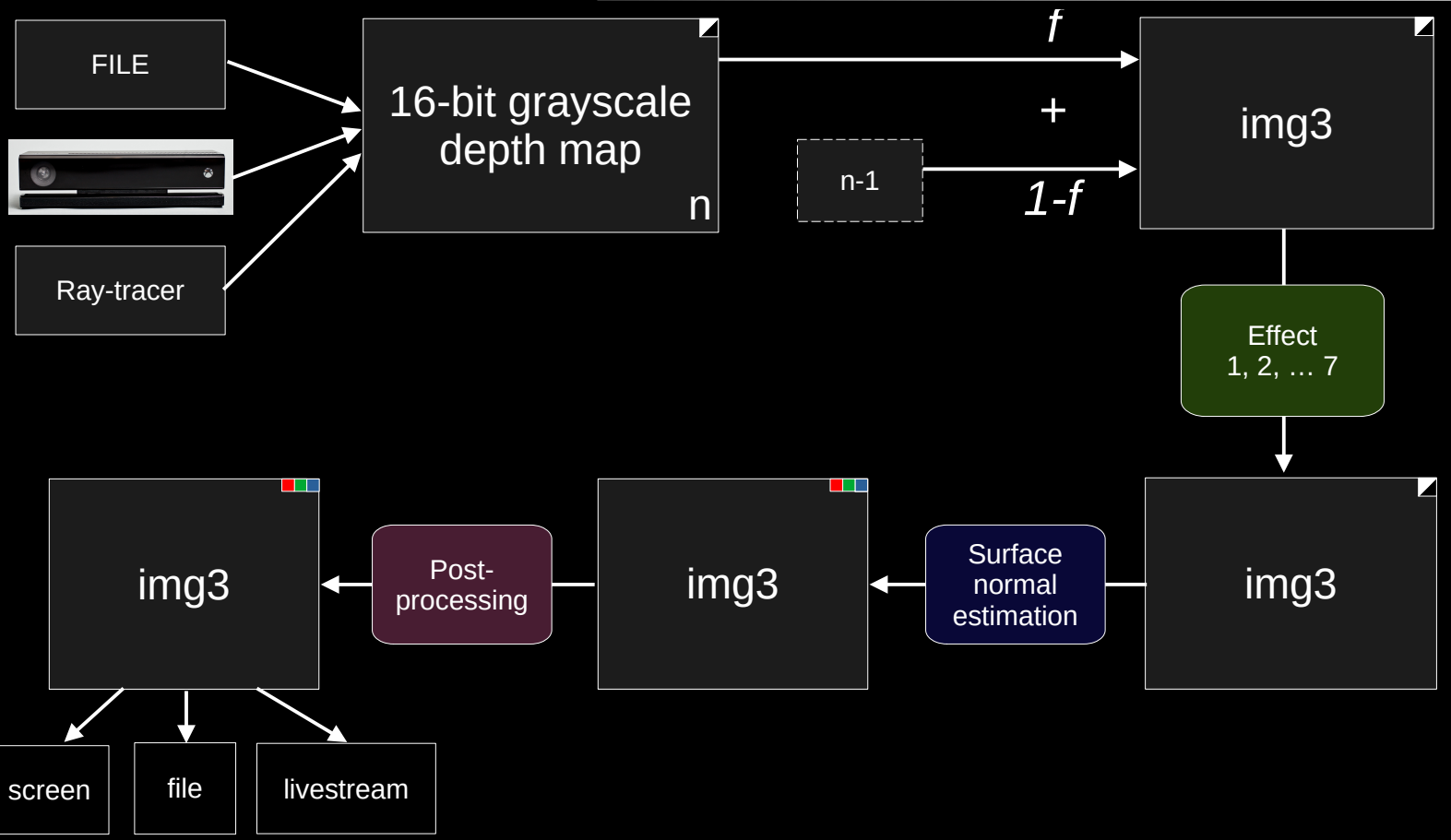
Intel / **TBB**

jeremycw / **httpserver.h**

Gamepad.h

OpenKinect/ **libfreenect2**

Pipeline


FILE
Ray-tracer
16-bit grayscale depth map
n
f
+
1-f
n-1
img3
Effect
1, 2, … 7
img3
Surface normal estimation
img3
Post-processing
img3
screen
file
livestream

Common Data Types

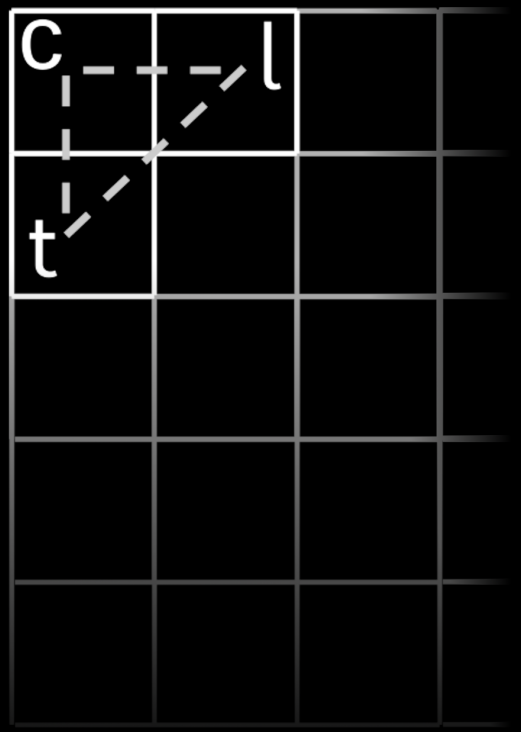
```
CV::Mat(int height, int width, int type)

CV::Vec3af(float a, float b, float c)

float pixels[width * height * sizeof(float)]
```

Raw frame

```
if (!listener.waitForNewFrame(frames, 10*1000)) // 10 seconds
{
»   std::cout << "timeout!" << std::endl;
»   if (REC)
»   »   vw.release();
}

framecount++;

libfreenect2::Frame *depth = frames[libfreenect2::Frame::Depth];

data = (char*)malloc(mys.width * mys.height * 4);
image = cvCreateImageHeader(cvSize(mys.width,mys.height), 32, 1);

Mat depth3(Size(512, 424), CV_32F, depth->data, Mat::AUTO_STEP);
flip(depth3, depth2, 1);

cvSetData(image, depth2.data, mys.width*4);
listener.release(frames);
```

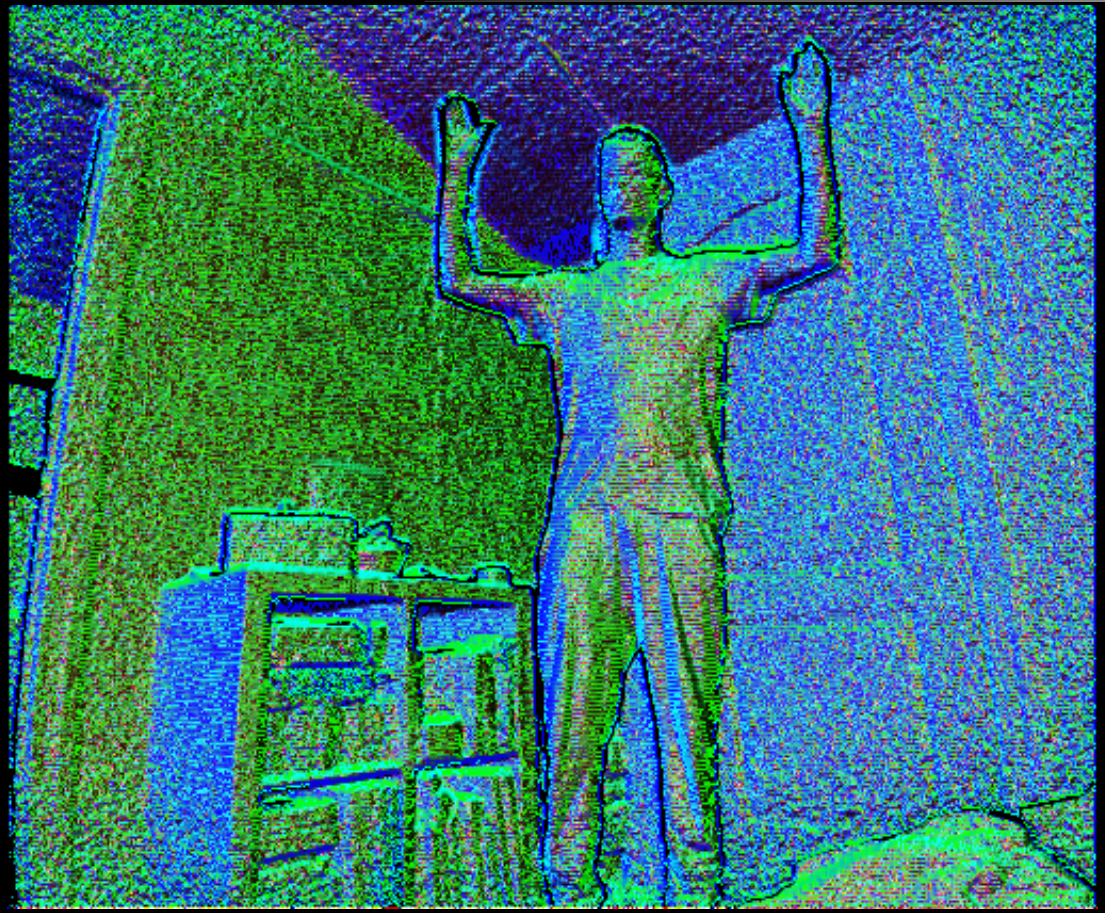
Raw frame

Surface Normal Estimation




```
parallel_for_(Range(0, (image4.rows)*image4.cols), [&](const Range& range){
    for (int r = range.start; r < range.end; r++)
    {
        int y = r / image4.cols;
        int x = r % image4.cols;

»   »   int rx = (x+mys.width+1)%mys.width;
»   »   int ry = (y+mys.height+1)%mys.height;

»   »   cv::Vec3d t(x, ry, image4.at<float>(ry,x));
»   »   cv::Vec3d l(rx, y, image4.at<float>(y,rx));
»   »   cv::Vec3d c(x, y, image4.at<float>(y,x));
»   »   cv::Vec3d d((l-c).cross(t-c));
»   »   cv::Vec3d n = cv::normalize(d).mul(vec_255);

»   »   if (n[2] != 255)
»   »   »   rgbMat.at<Vec3b>(y, x) = cv::Vec3b(n[0], n[1], n[2]);

»   »   else
»   »   »   rgbMat.at<Vec3b>(y, x) = cv::Vec3b(0, 0, 0);
    }
});
```

Surface Normal Estimation



Temporal Effects

```
addWeighted(image3, factor, image2, 1.f-factor, 0.0, image3);
```

1

```
cv::blur(image3, image3, cv::Size(bsize,bsize));
cv::bilateralFilter(image3, image2, a, bb, bb, BORDER_REPLICATE);
image2.copyTo(image4);
```

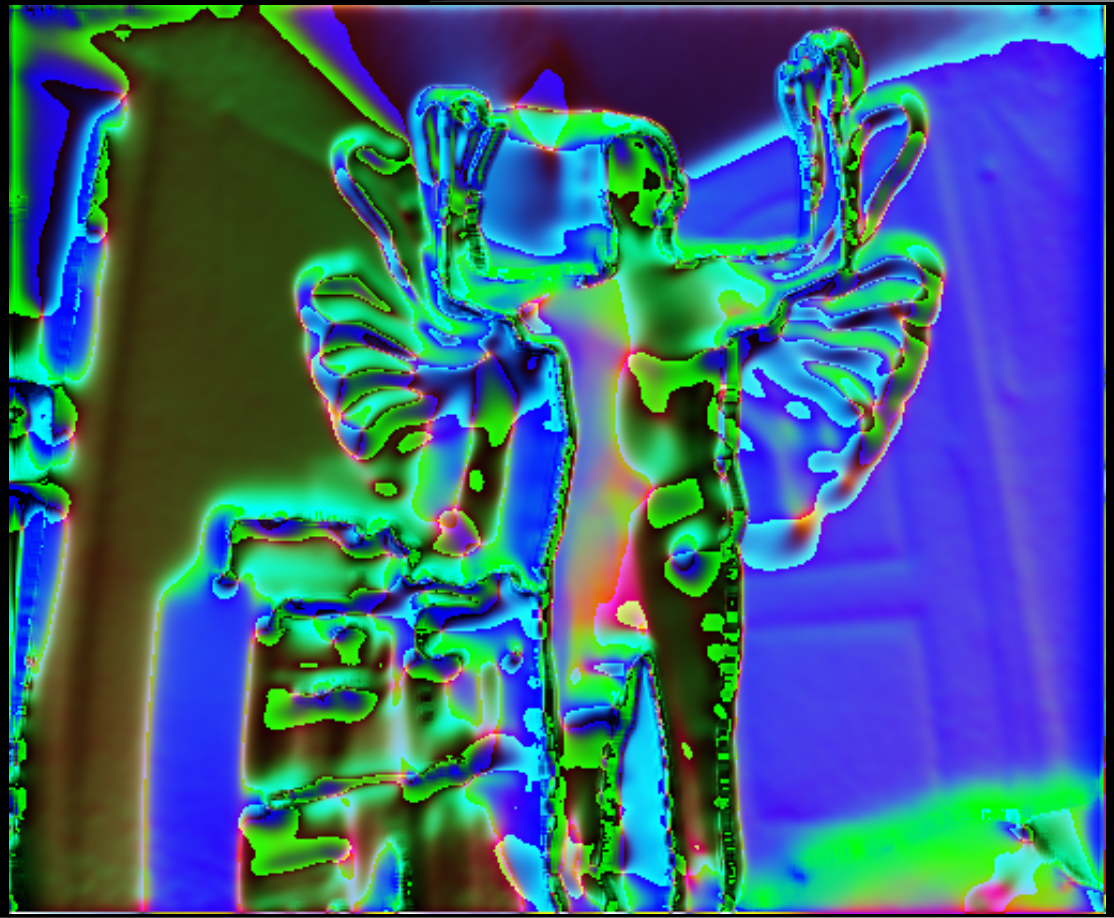
2

```
Mat kernel2 = Mat::ones(3, 3, CV_32F);
kernel2 = kernel2 / 9.f;
cv::dilate(image3, image4, kernel2, Point(-1, -1), 1, BORDER_REPLICATE, 1);
image4.copyTo(image3);
cv::bilateralFilter(image3, image2, a, bb, c, BORDER_ISOLATED);
image2.copyTo(image4);
```

4

```
Mat kernel4 = (Mat_<double>(3,3) << 2, 2, 0.5,
                                    0,  -1, 0.1,
                                    2, -0.5, -0.5);
filter2D(image3, image4, -1 , kernel4, Point(-1, -1), 0, BORDER_REFLECT);
image4.copyTo(image3);
cv::bilateralFilter(image3, image2, a, bb, bb, BORDER_ISOLATED);
```

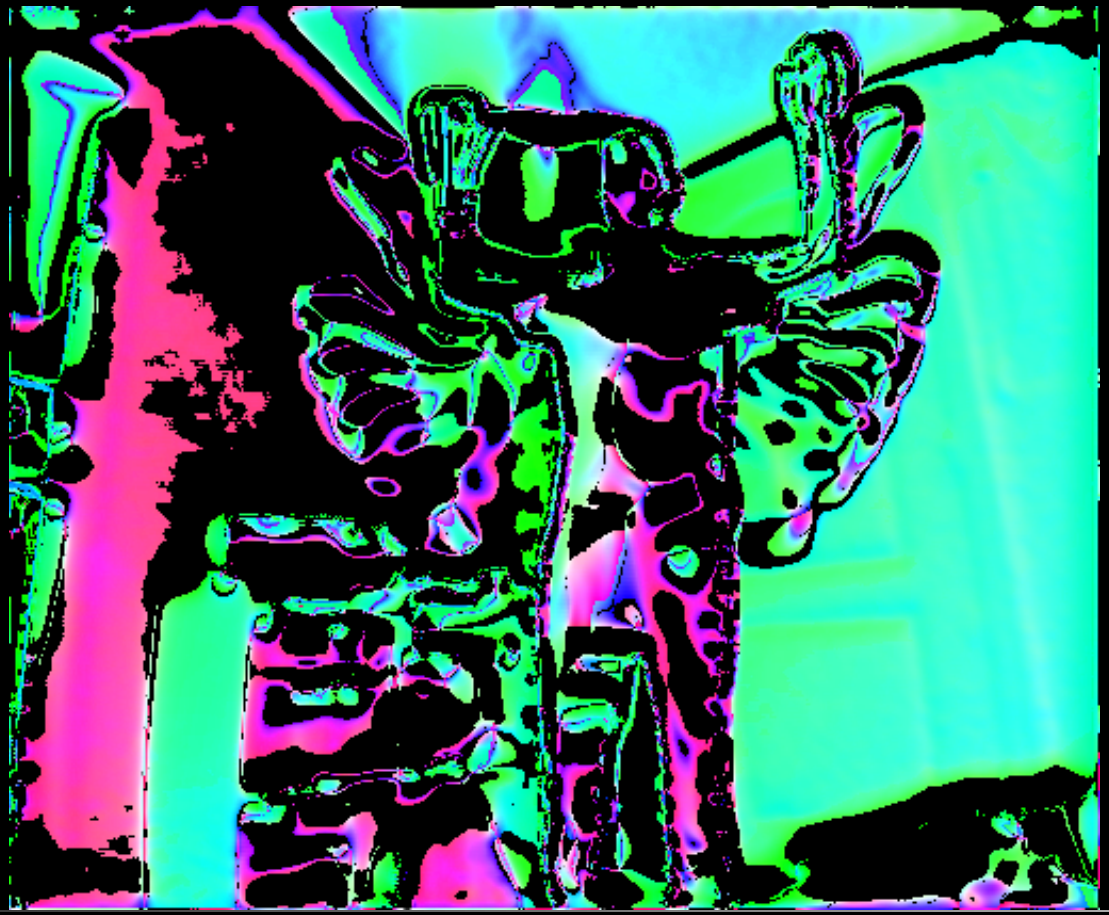
Temporal Effects



Post-Processing

```
cv::Mat *hsvImage = new Mat(mys.height, mys.width, CV_8UC3);
cvtColor(rgbMat, *hsvImage, COLOR_RGB2HSV_FULL, 3);

static vector<Mat> hsvChannels(3);
split(*hsvImage, hsvChannels);
vector<Mat> channels;

addWeighted(hsvChannels[0], 1.f-depth_factor, depthMat, depth_factor, 0.0, depthMat, CV_16U);

depthMat.forEach<uint16_t>([&](uint16_t &f, const int *position) {f = ((int)mys.hue_shift+f+((int)
((float)mcount*hue_shift_rate))) % 255;});

depthMat.convertTo(hsvChannels[0], CV_8U);

depthMat2.forEach<uint16_t>([&](uint16_t &f, const int *position) {
»   f = ( ( ((f + (mcount)) / v_1) % v_2 == 1) && (f > 2) ) ? f : 0 ;
});

depthMat2.convertTo(hsvChannels[2], CV_8U);
multiply(hsvChannels[2], hsvChannels[1], hsvChannels[2]);

merge(channels, rgbOut);
cvtColor(rgbOut, rgbOut, COLOR_HSV2RGB_FULL);
channels.clear();
```

Post-Processing



But can it XR?

Ray-tracing

```
else if (mys.ray_trace) {
    if (mcount % modval == 0) {
        rt->cam.fov = 70.f;

        /*embree::Vec3f to(0, 0, 0);
        rt->cam.from = from;*/

        if (!rt->rebuilding_scene) {
            active_buffer = !active_buffer;
            cout << "cam: " << rt->cam.str() << endl;
            memset(active_buffer ? df0[0] : df1[0], 0,
                   rt->mesh_height * rt->mesh_width * sizeof(float));

            rt->testEmbree(active_buffer ? df0 : df1, &active_buffer);
        }
    }
}
```

Ray-tracing - testEmbree

```
void RayTracer::testEmbree(float **depth_frames, bool *active_buf) {
»   embree::ISPCCamera ispc_cam = cam.getISPCCamera(mesh_width, mesh_height);

»   std::thread renderThread(renderFrameDispatch, depth_frames[0], mesh_width, mesh_height, 0.f,
    ispc_cam, this);
    renderThread.detach();
}
```

Ray-tracing 3 - renderFrameDispatch

```
void renderFrameDispatch (float* pixels,
                          const unsigned int width,
                          const unsigned int height,
                          const float time,
                          const embree::ISPCCamera& camera,
                           const RayTracer *rt)
{
»   const int numTilesX = (width +TILE_SIZE_X-1)/TILE_SIZE_X;
»   const int numTilesY = (height+TILE_SIZE_Y-1)/TILE_SIZE_Y;

»   const int totalThreads = 4;
»   std::vector<std::thread*> thre(totalThreads);

  for (int i = 0; i < totalThreads; i++) {
»   »   std::thread t1(renderFrameThread, pixels, width, height, time, camera, i, totalThreads, rt);
»   »   thre[i] = &t1;
»   »   thre[i]->detach();
»   }

}
```

Ray-tracing 4 - renderFrameThread

```
void renderFrameThread (float* pixels,
                        const unsigned int width,
                        const unsigned int height,
                        const float time,
                        const embree::ISPCCamera& camera,
»   »   »   »   »   »   int threadNum, const int totalThreads,
                       const RayTracer *rt)
{
  const int numTilesX = (width +TILE_SIZE_X-1)/TILE_SIZE_X;
  const int numTilesY = (height+TILE_SIZE_Y-1)/TILE_SIZE_Y;

  int totalTiles = numTilesX * numTilesY;
  int slice = totalTiles / totalThreads;

  for (int i = threadNum * slice ; i < std::min((threadNum+1) * slice, numTilesX * numTilesY); i++)
  {
»     renderTileTask((int)i, 0, pixels, width, height, time, camera, numTilesX, numTilesY, rt);
  }
}
```

Ray-tracing 5 - renderFrameTask

```
void renderTileTask (int taskIndex, int threadIndex, float* pixels,
                     const unsigned int width,
                     const unsigned int height,
                     const float time,
                     const embree::ISPCCamera& camera,
                     const int numTilesX,
                     const int numTilesY,
                   const RayTracer *rt)
{
  const unsigned int tileY = taskIndex / numTilesX;
  const unsigned int tileX = taskIndex - tileY * numTilesX;
  const unsigned int x0 = tileX * TILE_SIZE_X;
  const unsigned int x1 = std::min(x0+TILE_SIZE_X,width);
  const unsigned int y0 = tileY * TILE_SIZE_Y;
  const unsigned int y1 = std::min(y0+TILE_SIZE_Y,height);

  for (unsigned int y=y0; y<y1; y++) for (unsigned int x=x0; x<x1; x++)
  {
    renderPixelStandard(x, y, pixels, width, height, time, camera, rt);
  }
}
```

Ray-tracing 6 - renderPixelStandard

```
void renderPixelStandard(int x, int y,
                         float* pixels,
                         const unsigned int width,
                         const unsigned int height,
                         const float time,
                         const embree::ISPCCamera& camera,
                         const RayTracer *rt)
{
  Ray ray(embree::Vec3fa(camera.xfm.p),
»     »     embree::Vec3fa(normalize( x * camera.xfm.l.vx
»   »   »   »   »   »   »   »     + y * camera.xfm.l.vy
»   »   »   »   »   »   »   »     + camera.xfm.l.vz»)), 0.0f, embree::inf);

  rtcIntersect1(rt->scene, RTCRayHit_(ray));

  if (ray.geomID != RTC_INVALID_GEOMETRY_ID)
  {
»   pixels[y*width+x] = ray.tfar;
  }
}
```

Ray-tracing

Gstreamer setup

```
if (should_stream) {
»   ifstream inpipeline(pipeline_filename, ios::in | ios::binary);
»   string gstpipeline;
»   getline(inpipeline, gstpipeline);

»   cout << "pipeline: " << gstpipeline << endl;
»   vws = cv::VideoWriter(gstpipeline, cv::CAP_GSTREAMER, mys.target_fps, frame_size, true);

»   if (!vws.isOpened ()) {
    cout << "\n ***** Failed to open videowriter *****";
»   »   return -1;
»   }
}
```

```
if (should_stream) {
»   vws.write(rgbOut);
}
```

GStreamer pipeline - file

```
appsrc ! videoconvert ! video/x-raw,format=VUYA ! msdkh265enc rate-control=cqp qpi=20 qpp=20 qpb=20 gop-size=30 num-slices=1 ref-frames=1 b-frames=2 target-usage=3 hardware=true ! video/x-h265,profile=main-444 ! h265parse ! matroskamux ! filesink location=REC_NAME sync=false
```

GStreamer pipeline - stream

```
appsrc ! videoflip video-direction=2  !
videoconvert ! msdkh265enc rate-control=cqp
qpi=28 qpp=28 qpb=28 gop-size=30 num-
slices=1 ref-frames=1 b-frames=2 target-
usage=6 hardware=true ! video/x-
h265,profile=main ! h265parse ! rtph265pay
name=pay0 config-interval=-1 ! udpsink
host=192.168.1.178 port=8558
```

GStreamer pipeline - YouTube stream

```
flvmux name=mux streamable=true ! rtmpsink
location=rtmp://a.rtmp.youtube.com/live2/fef
t-3u6k-py7x-gkta-11p3  audiotestsrc volume=0
samplesperbuffer=44100 num-buffers=10 ! faac
! mux.  appsrc ! videoconvert ! vaapih264enc
rate-control=4 bitrate=12000 ! h264parse !
queue ! mux.
```

Demo time!