Pixvana SPIN Studio

Simplifying VR video creation and distribution

The Mediums of XR – Extended Reality

Cinema
Viewer as witness

”I saw”

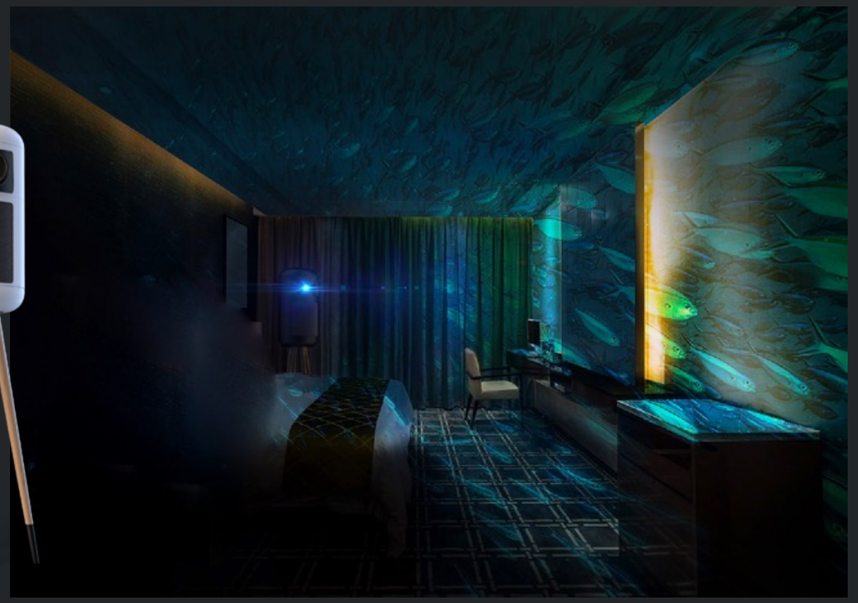
Virtual
Presence / immersion

“I experienced”

Augmented
Overlaid CGI
World-locked

“Context”

Mixed
Projected



“Holodeck”

Why Pixvana SPIN Studio?

Pixvana is a software startup building a video creation and delivery platform for the emerging mediums of virtual, augmented, and mixed reality (XR).

SPIN Studio makes it easy for anyone to shoot, edit and present 360 video content and deliver it from anywhere. Upload to the cloud, stitch content, create interactive stories and distribute (VRcast) content to any headset from one platform.

Who are Pixvana SPIN Studio customers?

VR Creatives

Small (1-10 person) production studios focus on VR tours and training

- VR tour creators take viewers to unique places or provide access to exclusive venues (exclusive real estate, Mt. Everest)
- Training creators use VR for company orientations, manufacturing demos, or sexual harassment prevention

Agencies + Media Companies

Both groups use our Pixvana Developer Kit (PDK) to distribute apps to end users

- Agencies create VR content to promote brands with a custom brand-centric app
- Media companies deliver content on their own infrastructure using Pixvana's OPF streaming format (CNN VR app on Oculus Rift)

Pixvana SPIN Studio features

Simplify the entire process of VR video creation and distribution with cloud-based tools

- **Library**: Upload and render up to 8Kx8K stereo video
- **Prep**: Transform raw camera footage into stitched 8K video
- **Story**: Tell complex stories with branching timelines, add interactive hotspots and graphics for immersive tours or training
- **Spin**: Send a playlist to groups of headsets for easy sharing of content (VRcasting)
- **Analytics:** Gain insights into viewer engagement

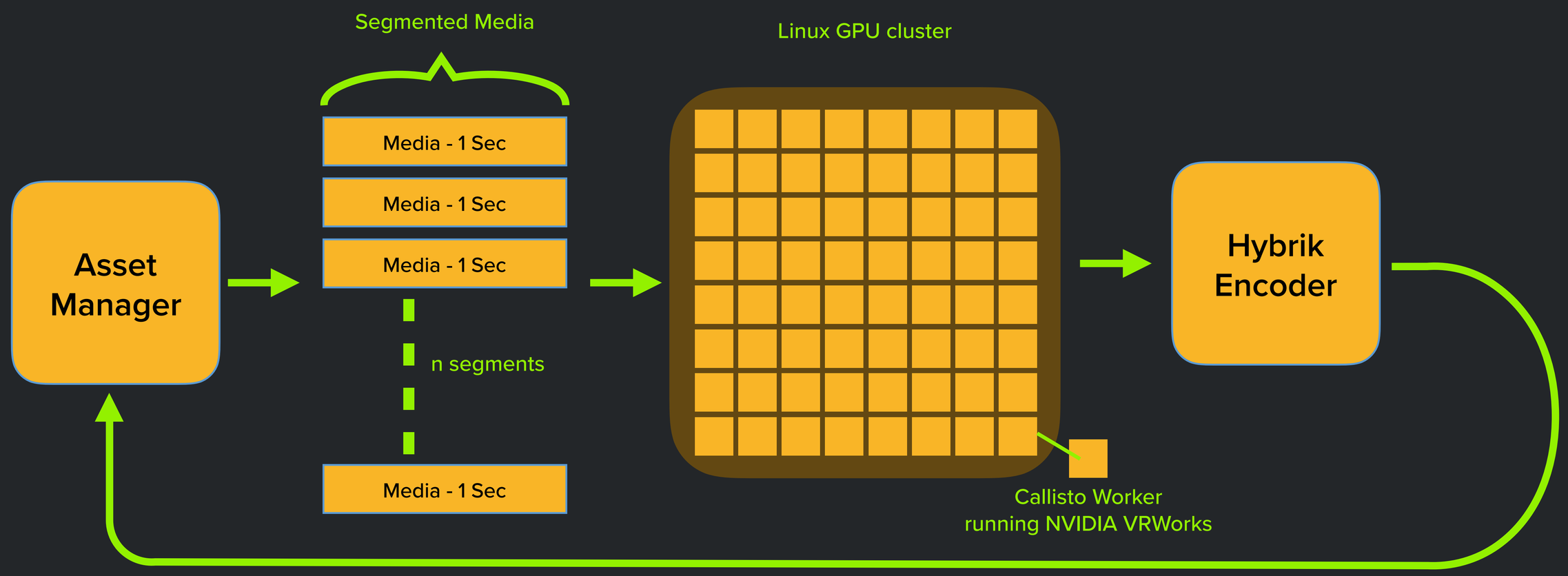
Prep: Transform camera footage into VR masters

SPIN Studio streamlines the entire process of turning camera sources into VR stories in just a few clicks.

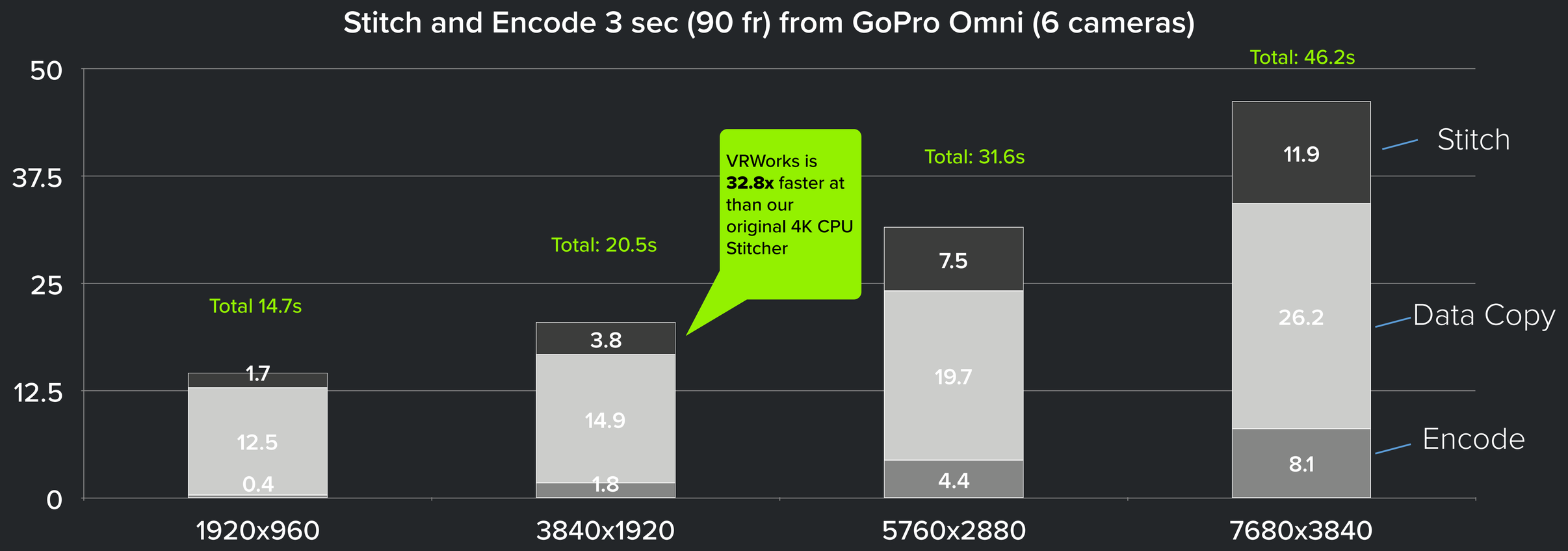
- Camera-agnostic stitching: Upload raw footage from a variety of popular camera rigs—GoPro Fusion/Omni, Insta360 Pro, Yi 360, Z CAM S1/K1/V1
- From mono 180 formats to stereo 360: we handle it all
- Straighten horizons with one click
- Color correction presets add saturation and contrast to every shot
- Built on the NVIDIA VRWorks 360 Video SDK

Pixvana SPIN Studio cloud processing

SPIN Studio stitches and encodes in parallel with up to 80 GPU-enabled machines running Linux on AWS/Azure.



NVIDIA VRWorks Stitching in Callisto



Frame Size	NVIDIA Stitcher (s)	Encode Frame(s)	Setup Time (s)	Total Time (s)	NVIDIA Stitch frame (ms)
2K	1.7	0.4	12.5	14.7	**19.3**
4K (UHD)	3.8	1.8	14.9	20.5	**42.7**
6K (5.7K)	7.5	4.4	19.7	31.6	**83.4**
8K (QUHD)	11.9	8.1	26.2	46.2	**132.0**

Technical Implementation

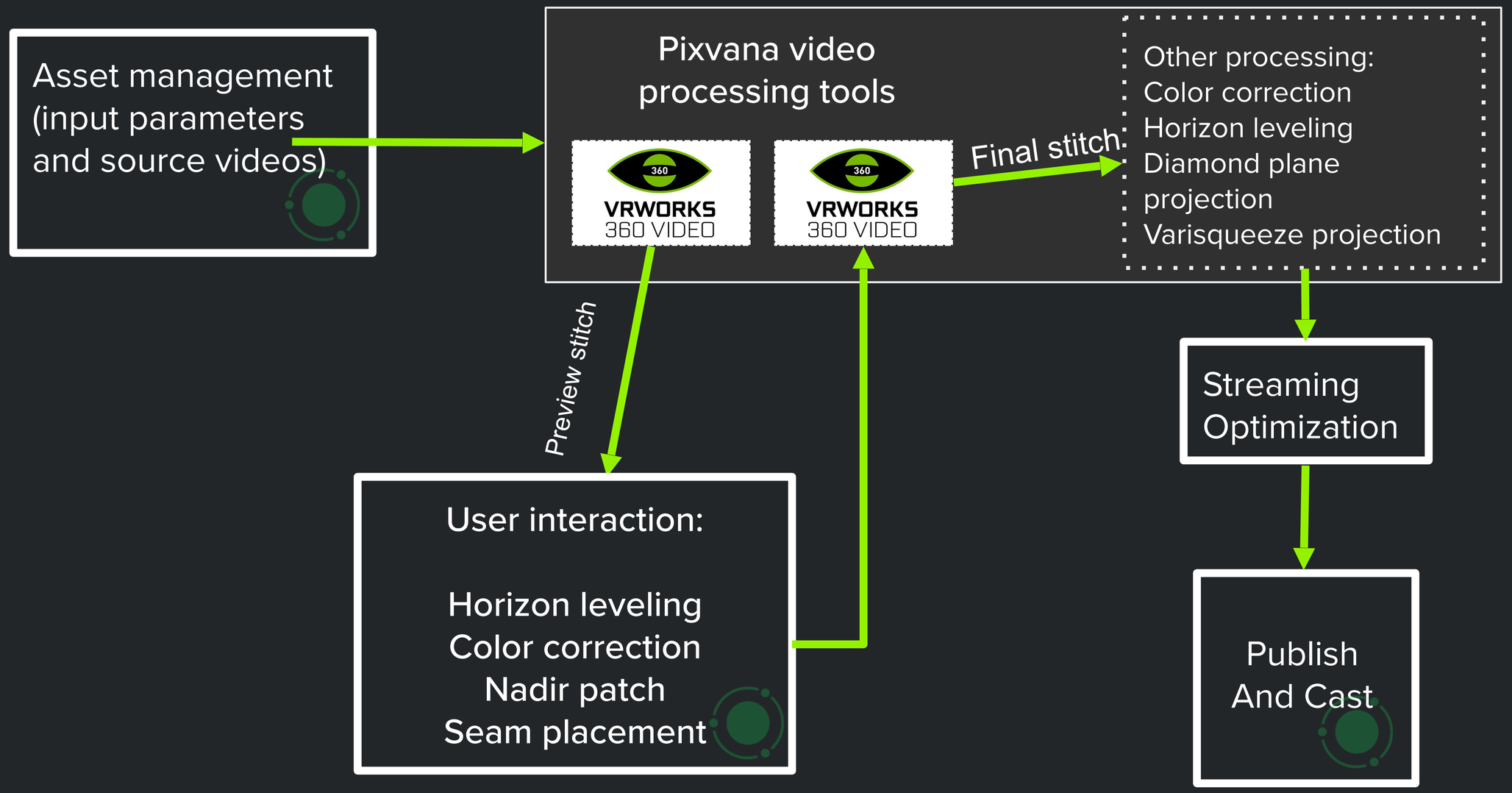
Dr. Paul Barsic, Lead Engineer, stitching

pixvana

VRWorks 360 Video SDK Key Features

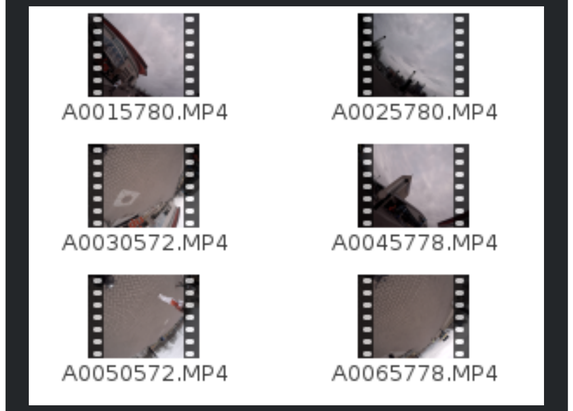
- ***GPU accelerated mono and stereo stitching***
- ***Automated calibration***
- Different input/output forms (***buffers***, files) are supported
- Stereo stitching performance scalability over multiple GPUs
- Hardware-accelerated decode and encode
- Blended audio output

Spin Studio stitching with VRWorks 360 Video



Pixvana Video Processing Tools

Sources



Parameters:
Output size
Stereo mode
Quality
etc.

VRWORKS 360 VIDEO

NVIDIA calibrator

NVIDIA Low Level Stitcher

Calibration File (XML)

Equirectangular stitch

Diamond plane & Varisqueeze projections

Bitrate ladder encoding

Seven Steps to Calibration

1. Create instance
2. Initialize camera rig
3. Set options
4. Pass image pointers
5. Perform calibration
6. Retrieve results
7. Destroy instance

```
// for now, this is 1, but it is possible to increase it
DEV_LOG("Creating calibration instance.");
verifyResult(
    nvcalibCreateInstance(framesCount, camCount, &hInstance),
    hInstance,
    "create calibration instance");

DEV_LOG("Setting rig properties.");
verifyResult(
    nvcalibSetRigProperties(hInstance, &calResult.videoRig_), hInstance, "set rig properties");

DEV_LOG("Setting calibration options.");
setCalibrationOptions(hInstance, calOptions);

DEV_LOG("Setting images.");
setCalibrationImages(hInstance, calResult.videoRig_, calImages);

DEV_LOG("Starting VRWorks calibration.");
verifyResult(nvcalibCalibrate(hInstance), hInstance, "VRWorks calibration");

DEV_LOG("Retrieving rig properties.");
verifyResult(
    nvcalibGetRigProperties(hInstance, &calResult.videoRig_),
    hInstance,
    "retrieve rig calibration properties");

verifyResult(
    nvcalibDestroyInstance(hInstance), hInstance, "destroy VRWorks calibration instance");
```

Calibration step 3

Set the calibration parameters

Calibration instance

Description of input imagery

```
        if((res = nvcalibSetCameraProperty(
                hCalibration,
                camIndex,
                nvcalibCameraProperties::NVCALIB_CAM_PROP_INPUT_IMAGE_PITCH,
                nvcalibDataType::NVCALIB_DATATYPE_UINT32,
                1,
                (void*)(&pitch))) != nvcalibResult::NVCALIB_SUCCESS)
        {
            throwVRWorksError(
                "Error setting input image pitch.", getErrorString(res, hCalibration));
        }
        imagePtrs[camIndex] = currImgPtr;
    }
```

Calibration step 4

Pass pointers to the source images

Calibration instance

Source image pointers

```
    // Add all images
    DEV_LOG("Set images");
    if((res = nvcalibSetImages(hCalibration, (const void**)imagePtrs.data())) !=
       nvcalibResult::NVCALIB_SUCCESS)
    {
        throwVRWorksError(
            "Error setting calibration frame.", getErrorString(res, hCalibration));
    }
}
```

Stitching

1. Load calibration
2. Initialize stitcher
3. Load images
4. Call the stitcher
5. Get stitched equirect
6. Destroy instance

```
interpretCameraRigXml(calParamsXml, calParams.cameraProperties_, &calParams.videoRig_, YDirectionStr);
```

```
DEV_LOG("initialize stitcher instance.");
nvssVideoHandle stitcher;
nvssVideoCreateInstance(&stitcher_props, &calParams.videoRig_, &stitcher);

DEV_LOG("load the input data onto the GPU");
loadImages2GPU(stitcher, calParams.videoRig_, inputImages);

cudaStreamSynchronize(cudaStreamDefault);

DEV_LOG("Calling nvssVideoStitch");
nvssVideoStitch(stitcher);

cudaStreamSynchronize(cudaStreamDefault);

DEV_LOG("coying output panorama from CUDA buffer.");
if(cudaMemcpy2D( out_stacked , output_image.row_bytes, output_image.dev_ptr, output_image.pitch,
            output_image.row_bytes, output_image.height, cudaMemcpyDeviceToHost) != cudaSuccess)
{
    throwVRWorksError("Error copying output stacked panorama from CUDA buffer");
}

// Clean up
nvssVideoDestroyInstance(stitcher);
```

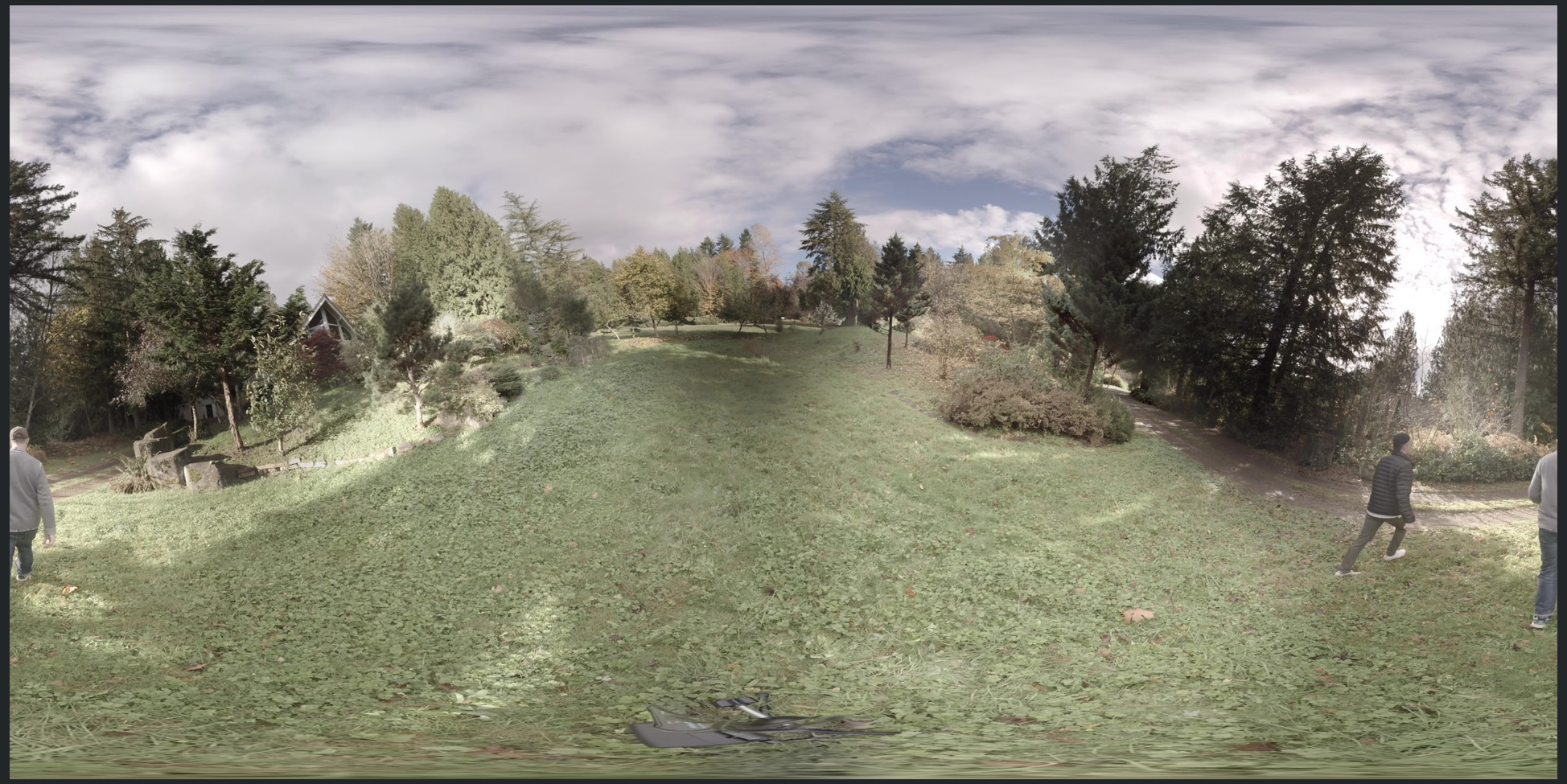
Commentary on step 3: loadImages2GPU

loadImages2GPU is a wrapper around cudaMemcpy2D:

1. Reserve a buffer on the GPU for each data source
2. Convert RGB images to RGBA
3. Call cudaMemcopy2D

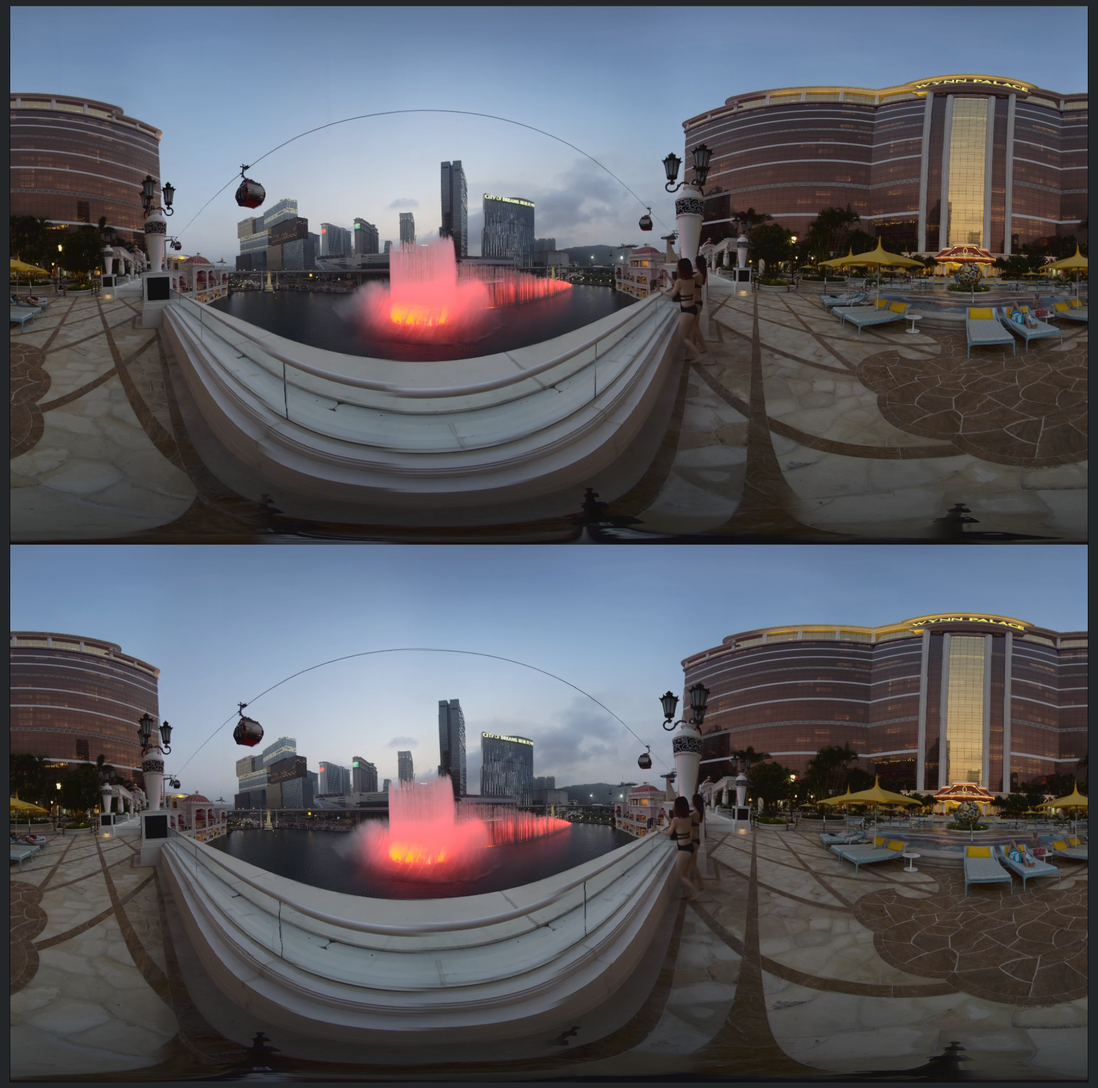
Working to optimize this step to eliminate copy and swizzle

Stitched output sample (GoPro Omni, mono)



The stitched output requires adjustment.

Stitched output example (Z Cam V1 Pro, stereo)



Successful Results with Different Camera Rigs

GoPro OMNI
mono

Insta360 Pro
mono

Z CAM V1 Pro
mono
stereo

VRWorks 360 Video Enables Stitching

Automatic calibration allows Pixvana to import video from any 360° camera rig.

Existing data flow did not change: VRWorks uses CUDA buffers.

Output image size is limited by system resources .

Linux SDK allows scalable cloud deployment.

Computational stereo processing for equatorial camera arrays.

VRWorks 360 Video Challenges

Up front calibration is time consuming and sensitive to input quality.

When calibration fails, it is difficult to diagnose calibration errors.

Seam placement is automatically determined, great for speed but doesn’t enable post-stitch refinement

Exposure compensation does not allow for user input.