Porting pbrt to the GPU While Preserving its Soul

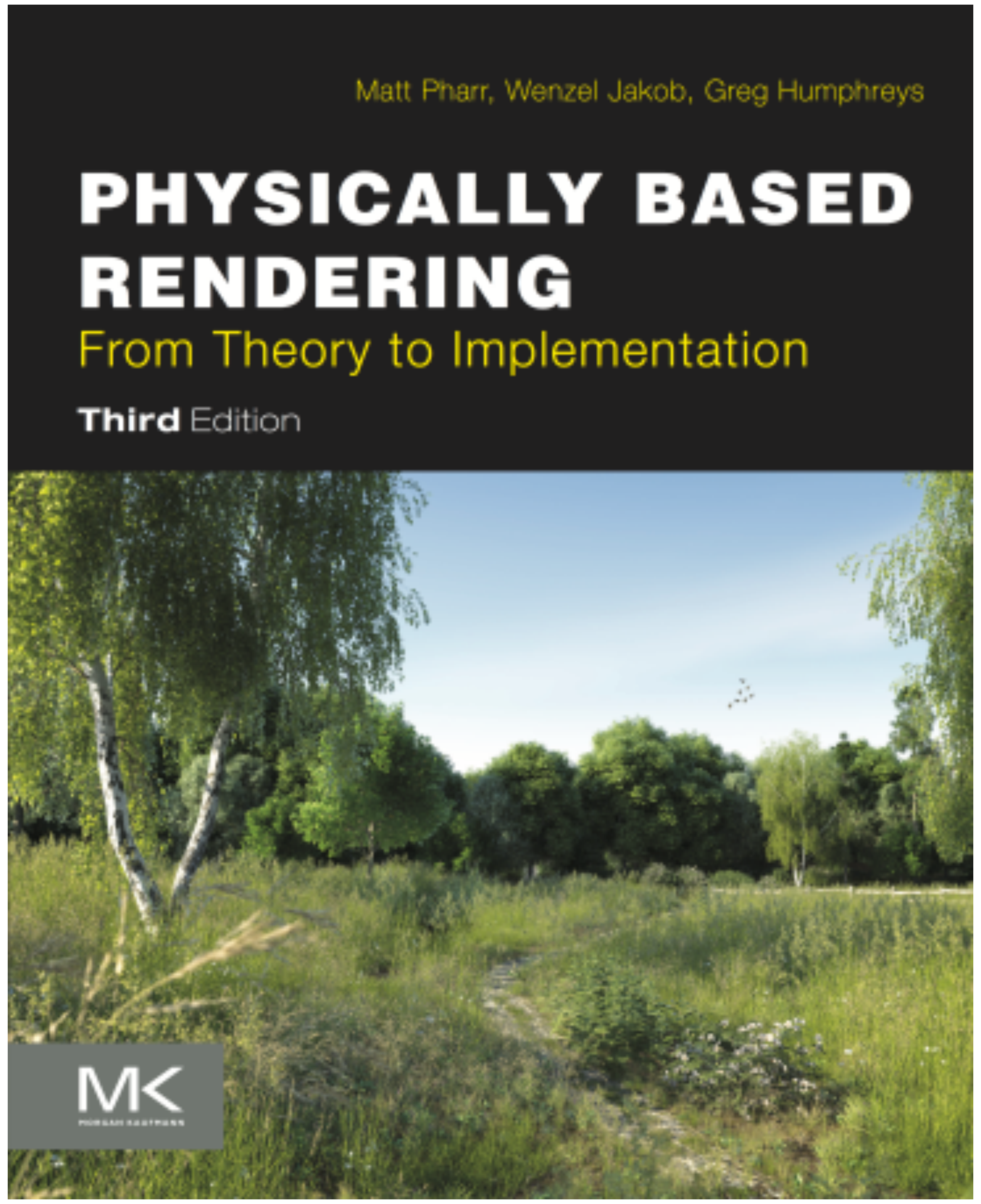
Matt Pharr
NVIDIA

15 July 2020

pbrt at HPG???

pbrt Background

- Ray-tracer implemented as a literate program
- Book goes all the way from equations / ideas to C++ code
- Book: ~1000 pages
- Renderer: ~72k LOC, C++
- First edition in 2004, some code dates to 1998


Matt Pharr, Wenzel Jakob, Greg Humphreys
PHYSICALLY BASED RENDERING
From Theory to Implementation
Third Edition
MK

pbrt Context / Constraints

- System's goals are primarily pedagogical
    - Value proposition: C++ and calculus are the only prerequisites
- Try to be relevant for 5-10 years
    - Avoid external APIs (beyond the stdlib)
- Portability is important

Tension: Performance vs. Clarity

- Want to teach something about system organization and design
  - Performance is a big part of rendering
  - But maximizing performance can get grungy…
- Example: pbrt is multi-threaded–can discuss mutual exclusion, atomics, false sharing, …

pbrt’s Ray-AABB Intersection Code

```
template <typename T>
inline bool Bounds3<T>::IntersectP(const Point3f &o, const Vector3f &d, Float tMax,
                                   Float *hitt0, Float *hitt1) const {
    Float t0 = 0, t1 = tMax;
    for (int i = 0; i < 3; ++i) {
        // Update interval for _i_th bounding box slab
        Float invRayDir = 1 / d[i];
        Float tNear = (pMin[i] - o[i]) * invRayDir;
        Float tFar = (pMax[i] - o[i]) * invRayDir;

        // Update parametric interval from slab intersection $t$ values
        if (tNear > tFar)
            std::swap(tNear, tFar);

        // Update _tFar_ to ensure robust ray--bounds intersection
        tFar *= 1 + 2 * gamma(3);
        t0 = tNear > t0 ? tNear : t0;
        t1 = tFar < t1 ? tFar : t1;
        if (t0 > t1)
            return false;
    }
    if (hitt0) *hitt0 = t0;
    if (hitt1) *hitt1 = t1;
    return true;
}
```

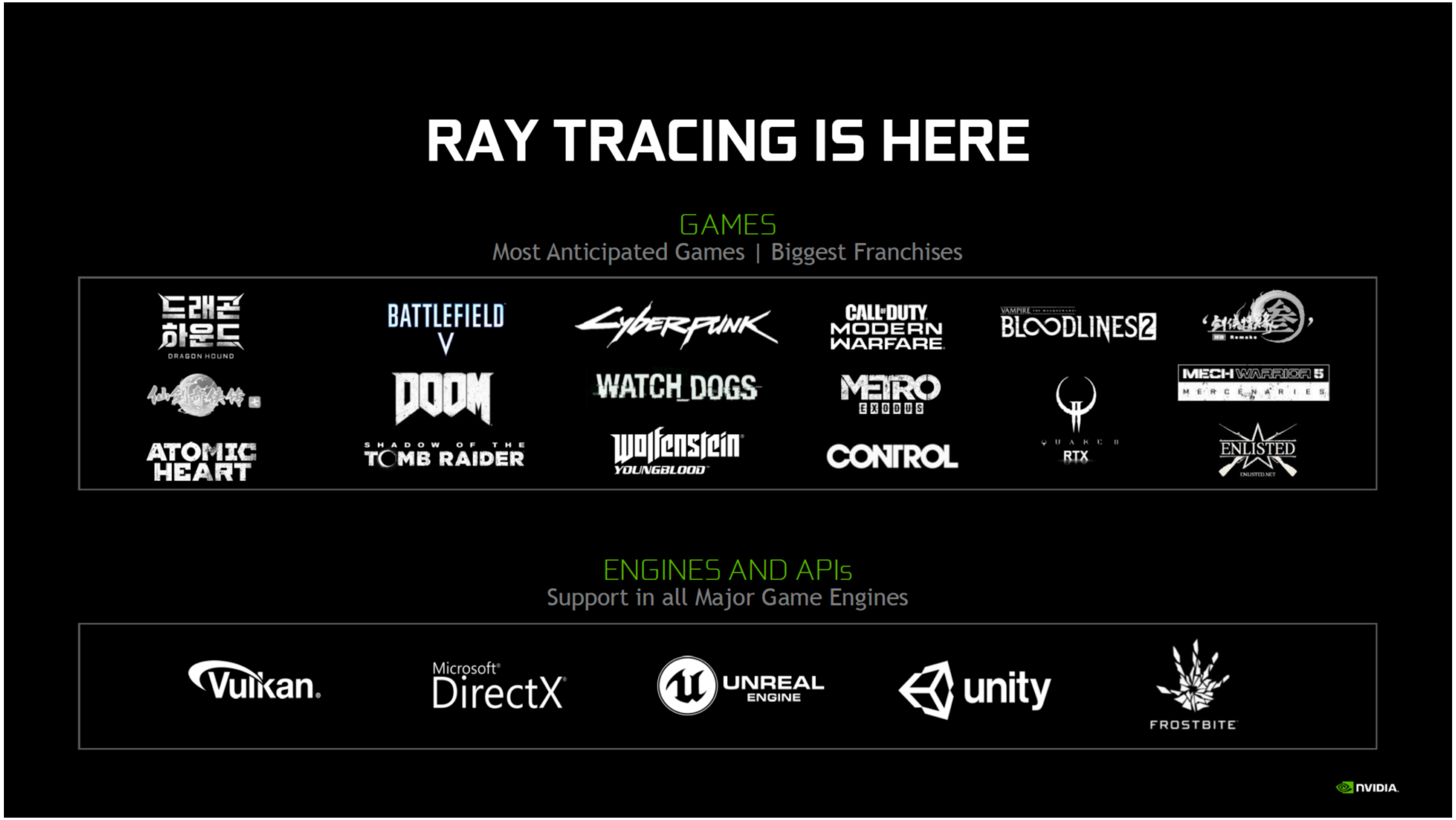
Not pbrt’s Ray-AABB Intersection Code

```
static bool ray_box(const Bounds3f &box, const Ray &ray, float *tMin, float *tMax) {
    const __m128 plus_inf = _mm_load_ps((const float *const)(ps_cst_plus_inf));
    const __m128 minus_inf = _mm_load_ps((const float *const)(ps_cst_minus_inf));
    const __m128 box_min = _mm_load_ps((const float *const)(&box.pMin));
    const __m128 box_max = _mm_load_ps((const float *const)(&box.pMax));
    const __m128 pos = _mm_load_ps((const float *const)&ray.o);
    const __m128 inv_dir = _mm_load_ps((const float *const)(&ray.inv_dir));
    const __m128 l1 = _mm_mul_ps(_mm_sub_ps(box_min, pos), inv_dir);
    const __m128 l2 = _mm_mul_ps(_mm_sub_ps(box_max, pos), inv_dir);
    const __m128 filtered_l1a = _mm_min_ps(l1, plus_inf);
    const __m128 filtered_l2a = _mm_min_ps(l2, plus_inf);
    const __m128 filtered_l1b = _mm_max_ps(l1, minus_inf);
    const __m128 filtered_l2b = _mm_max_ps(l2, minus_inf);
    __m128 lmax = _mm_max_ps(filtered_l1a, filtered_l2a);
    __m128 lmin = _mm_min_ps(filtered_l1b, filtered_l2b);
    const __m128 lmax0 = _mm_shuffle_ps(lmax, lmax, 0x39);
    const __m128 lmin0 = _mm_shuffle_ps(lmin, lmin, 0x39);
    lmax = _mm_min_ss(lmax, lmax0);
    lmin = _mm_max_ss(lmin, lmin0);
    const __m128 lmax1 = _mm_movehl_ps((lmax), (lmax));
    const __m128 lmin1 = _mm_movehl_ps((lmin), (lmin));
    lmax = _mm_min_ss(lmax, lmax1);
    lmin = _mm_max_ss(lmin, lmin1);
    const bool ret =
        _mm_comige_ss(lmax, _mm_setzero_ps()) & _mm_comige_ss(lmax, lmin);
    _mm_store_ss((float *const)&tMin, lmin);
    _mm_store_ss((float *const)&tMax, lmax);
    return ret;
}
```

pbrt $\cap$ ispc $= \emptyset$

- Though based on C, ispc is a new language
  - It’s too much to require learning a new language to read the book…
- But yet…
  - SIMD is important for CPU production rendering
  - Would like to discuss ray packets, multi-BVHs, sorting for shading…

"Try to be relevant…"



RAY TRACING IS HERE

GAMES

Most Anticipated Games | Biggest Franchises

드래곤 하운드 DRAGON HOUND
BATTLEFIELD V
Cyberpunk
CALL OF DUTY MODERN WARFARE
VAMPIRE BLOODLINES 2
MECHWARRIOR 5 MERCENARIES
DOOM
WATCH_DOGS
METRO EXODUS
ATOMIC HEART
SHADOW OF THE TOMB RAIDER
Wolfenstein YOUNGBLOOD
CONTROL
QUAKE II RTX
ENLISTED

ENGINES AND APIs

Support in all Major Game Engines

Vulkan.
Microsoft DirectX
UNREAL ENGINE
unity
FROSTBITE
NVIDIA

Porting Approach

- CUDA + OptiX or bust
  - CUDA: only option given C++ and portability requirements
    - Prospect of maximizing shared code between CPU and GPU
  - OptiX: GPU-accelerated intersection tests
    - And can side-step explaining highly-parallel creation of BVHs, …

Porting Approach

- CUDA + OptiX or bust
  - CUDA: only option given C++ and portability requirements
    - Prospect of maxi U and GPU
  - OptiX: GPU-accelerated intersection tests
    - And can side-step explaining highly-parallel creation of BVHs, …

Ray intersection is a black-box for the GPU path. But still have all the details for the CPU…

Porting Approach

- CUDA + OptiX or bust
- GPU path as alternative to CPU, not replacement
- Fail fast: is it going to work in the first place?
  - (Work == doesn't complexify code excessively + perf. is decent)

➡Start making pictures ASAP

電
ШММБ

Crossing The Chasm

- Extensive `__host__ __device__` annotations…
- Data structure initialization all CPU-side, like before
  - Ubiquitous plumbing of `std::pmr::polymorphic_allocator`
- `GPUParallelFor` + `__device__` lambda functions
- Tagged-dispatch in place of virtual function calls

Memory Allocations

```
using Allocator = std::pmr::polymorphic_allocator<std::byte>;

class PiecewiseConstant1D {
  PiecewiseConstant1D(std::vector<Float> f, Allocator alloc = {})
    : func(f.begin(), f.end(), alloc), cdf(f.size() + 1, alloc) {
    // Compute integral of step function at $x_i$
    cdf[0] = 0;
    size_t n = f.size();
    for (size_t i = 1; i < n + 1; ++i)
      cdf[i] = cdf[i - 1] + func[i - 1] / n;
    …
  }

  …
  pstd::vector<Float> func, cdf;
};
```

Memory Allocations

```
using Allocator = std::pmr::polymorphic_allocator<std::byte>;

class PiecewiseConstant1D {
  PiecewiseConstant1D(std::vector<Float> f, Allocator alloc = {})
    : func(f.begin(), f.end(), alloc), cdf(f.size() + 1, alloc) {
    // Compute integral of step function at $x_i$
    cdf[0] = 0;
    size_t n = f.size();
    for (size_t i = 1; i < n + 1; ++i)
      cdf[i] = cdf[i - 1] + func[i - 1] / n;
    …
  }

  …
  pstd::vector<Float> func, cdf;
};
```

Pass allocator that allocates unified memory for GPU rendering…

GPU Kernel Launch

```
PathState pathState[NumPixels];
FilmHandle film;
// …

GPUParallelFor("Update Film", pixelsPerPass,
  [=] PBRT_GPU (PixelIndex pixelIndex) {
    const PathState &pathState = pathStates[pixelIndex];
    Point2i pPixel = pathState.pPixel;
    if (!InsideExclusive(pPixel, film.PixelBounds()))
        return;

    SampledSpectrum L = pathState.L * pathState.cameraWeight;
    film.AddSample(pPixel, L, pathState.filterWeight);
});
```

GPU Kernel Launch

```
PathState pathState[NumPixels];
FilmHandle film;
// …

GPUParallelFor("Update Film", pixelsPerPass,
  [=] PBRT_GPU (PixelIndex pixelIndex) {
    const PathState &pathState = pathStates[pixelIndex];
    Point2i pPixel = pathState.pPixel;
    if (!InsideExclusive(pPixel, film.PixelBounds()))
        return;

    SampledSpectrum L = pathState.L * pathState.cameraWeight;
    film.AddSample(pPixel, L, pathState.filterWeight);
});
```

Virtual Functions →Tagged Dispatch

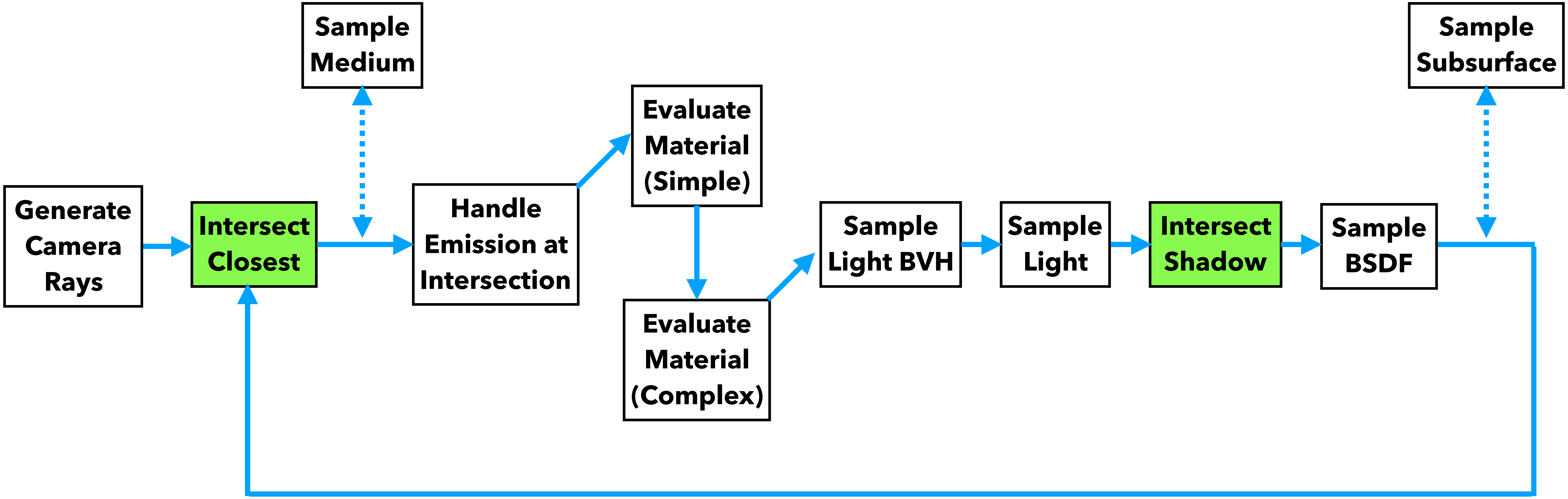
```
class CameraHandle :
    public TaggedPointer<PerspectiveCamera, OrthographicCamera,
                         SphericalCamera, RealisticCamera> {
public:
  PBRT_CPU_GPU
  pstd::optional<CameraRay> GenerateRay(const CameraSample &sample,
                                        const SampledWavelengths &lambda) const {
    switch (Tag()) {
    case TypeIndex<PerspectiveCamera>():
      return Cast<PerspectiveCamera>()->GenerateRay(sample, lambda);
    case TypeIndex<OrthographicCamera>():
      return Cast<OrthographicCamera>()->GenerateRay(sample, lambda);
    case TypeIndex<SphericalCamera>():
      return Cast<SphericalCamera>()->GenerateRay(sample, lambda);
    case TypeIndex<RealisticCamera>():
      return Cast<RealisticCamera>()->GenerateRay(sample, lambda);
    }
}
```

(TaggedPointer builds on DiscriminatedPtr from Facebook's folly library)

Tagged Dispatch v2

```
class CameraHandle :
    public TaggedPointer<PerspectiveCamera, OrthographicCamera,
                         SphericalCamera, RealisticCamera> {
public:
  PBRT_CPU_GPU
  pstd::optional<CameraRay> GenerateRay(const CameraSample &sample,
                                        const SampledWavelengths &lambda) const {
    auto generateRay = [&](auto ptr) -> pstd::optional<CameraRay> {
      return ptr->GenerateRay(sample, lambda);
    };
    return Apply(generateRay);
  }
```

Path-Tracing Pipeline



Generate Camera Rays

Intersect Closest

Sample Medium

Handle Emission at Intersection

Evaluate Material (Simple)

Evaluate Material (Complex)

Sample Light BVH

Sample Light

Intersect Shadow

Sample BSDF

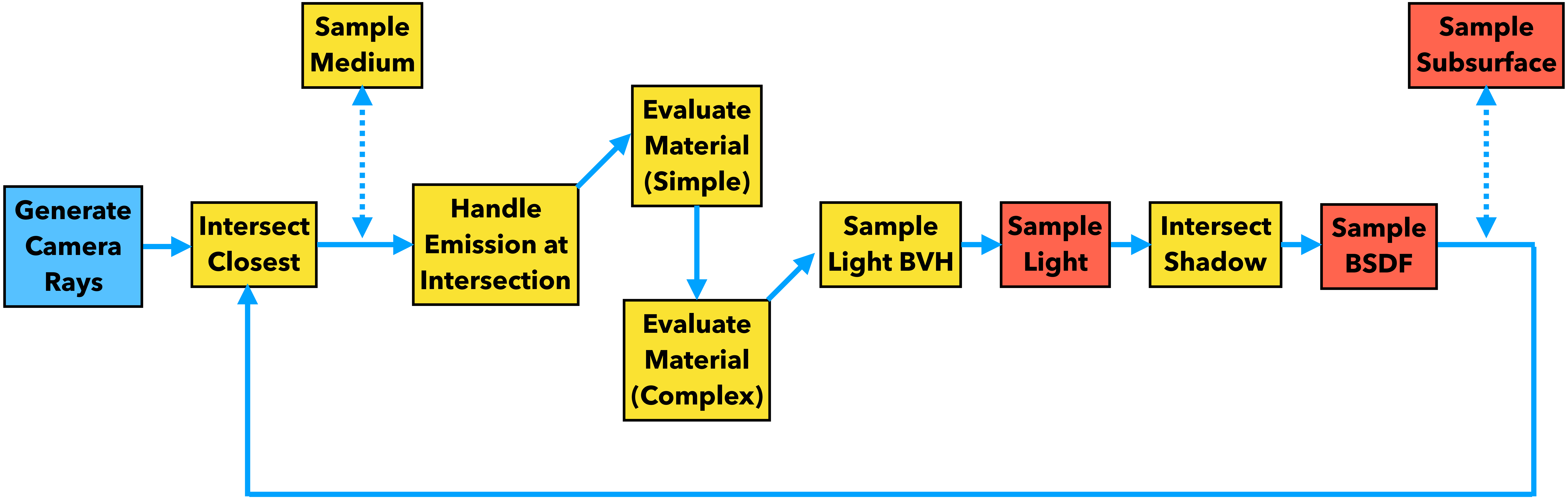
Sample Subsurface

OptiX

OptiX

HPG 2020

Parallelism Domains: Maximize Control Convergence

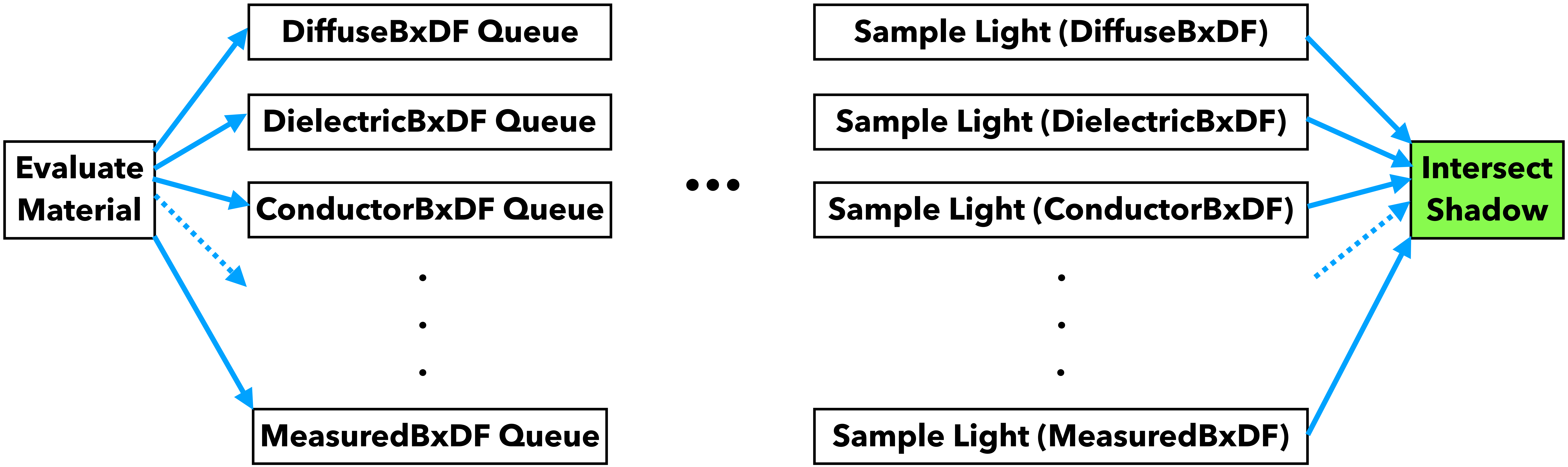


For each Pixel

For each Ray

For each BxDF type, For Each Ray

BxDF Sorting


Evaluate Material
DiffuseBxDF Queue
DielectricBxDF Queue
ConductorBxDF Queue
MeasuredBxDF Queue
...
Sample Light (DiffuseBxDF)
Sample Light (DielectricBxDF)
Sample Light (ConductorBxDF)
Sample Light (MeasuredBxDF)
Intersect Shadow

Resulting improved control convergence gave ~2x speedup (overall) on San Miguel

Performance vs. CPU pbrt

(RTX2080 vs 6c/12t @ 3.4GHz)

51x

53x

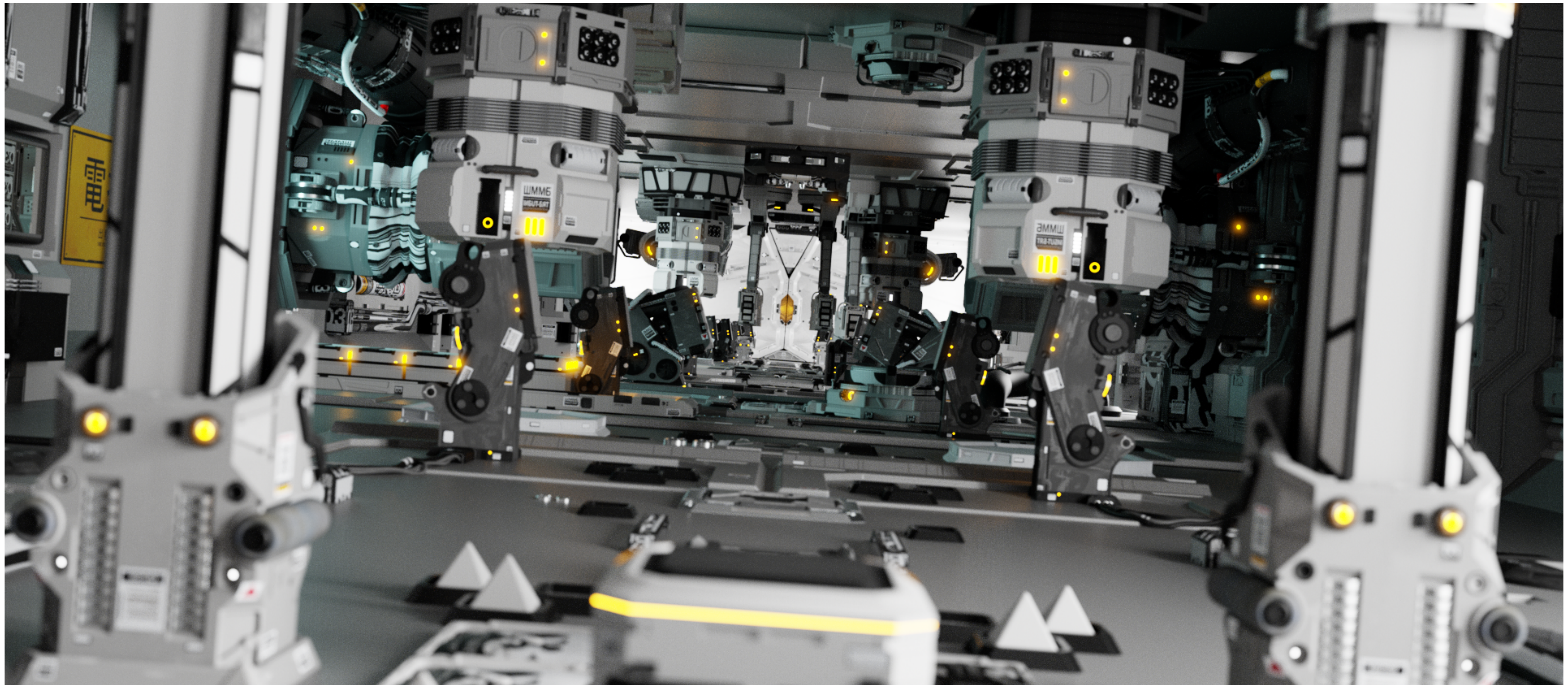
30x

32x

27x

28x

Performance vs. Optimized DX12 RT *



~1 order of magnitude slower

* (Not an exact apples-to-apples to comparison)

Demo interlude…

Performance Breakdown: San Miguel @ 1080p, 1spp

```
Reset sampler dimension                             2 launches       3.72 ms /   3.1%
Generate Camera rays                                2 launches       9.72 ms /   8.1%
Initialize PathState                                2 launches       2.71 ms /   2.3%
Clear intersections                                 6 launches       5.82 ms /   4.8%
Path tracing closest hit rays                       6 launches      32.97 ms /  27.4%
Handle ray-found emission                           6 launches       2.65 ms /   2.2%
Bump and Material::GetBSDF/GetBSSRDF                4 launches      17.60 ms /  14.6%
Bump and Material::GetBSDF/GetBSSRDF                4 launches       3.98 ms /   3.3%
Choose Light to Sample                              4 launches       7.59 ms /   6.3%
Sample direct - DiffuseBxDF                         4 launches       9.85 ms /   8.2%
Sample direct - CoatedDiffuseBxDF                   4 launches       2.83 ms /   2.4%
Path tracing shadow rays                            4 launches       6.84 ms /   5.7%
Sample indirect - DiffuseBxDF                       4 launches       7.36 ms /   6.1%
Sample indirect - CoatedDiffuseBxDF                 4 launches       2.07 ms /   1.7%
Sample indirect - DielectricInterfaceBxDF           4 launches       0.69 ms /   0.6%
Update Film                                         2 launches       1.98 ms /   1.6%
Other                                              86 launches       1.83 ms /   1.5%
```

Performance Breakdown: San Miguel @ 1080p, 1spp

```
Reset sampler dimension                            2 launches      3.72 ms /   3.1%
Generate Camera rays                               2 launches      9.72 ms /   8.1%
Initialize PathState                               2 launches      2.71 ms /   2.3%
Clear intersections                                6 launches      5.82 ms /   4.8%
Path tracing closest hit rays                      6 launches     32.97 ms /  27.4%
Handle ray-found emission                          6 launches      2.65 ms /   2.2%
Bump and Material::GetBSDF/GetBSSRDF               4 launches     17.60 ms /  14.6%
Bump and Material::GetBSDF/GetBSSRDF               4 launches      3.98 ms /   3.3%
Choose Light to Sample                             4 launches      7.59 ms /   6.3%
Sample direct - DiffuseBxDF                        4 launches      9.85 ms /   8.2%
Sample direct - CoatedDiffuseBxDF                  4 launches      2.83 ms /   2.4%
Path tracing shadow rays                           4 launches      6.84 ms /   5.7%
Sample indirect - DiffuseBxDF                      4 launches      7.36 ms /   6.1%
Sample indirect - CoatedDiffuseBxDF                4 launches      2.07 ms /   1.7%
Sample indirect - DielectricInterfaceBxDF          4 launches      0.69 ms /   0.6%
Update Film                                        2 launches      1.98 ms /   1.6%
Other                                             86 launches      1.83 ms /   1.5%
```

Performance Breakdown:
San Miguel @ 1080p, 1spp

```
Reset sampler dimension                           2 launches      3.72 ms /   3.1%
Generate Camera rays                              2 launches      9.72 ms /   8.1%
Initialize PathState                              2 launches      2.71 ms /   2.3%
Clear intersections                               6 launches      5.82 ms /   4.8%
Path tracing closest hit rays                     6 launches     32.97 ms /  27.4%
Handle ray-found emission                         6 launches      2.65 ms /   2.2%
Bump and Material::GetBSDF/GetBSSRDF              4 launches     17.60 ms /  14.6%
Bump and Material::GetBSDF/GetBSSRDF              4 launches      3.98 ms /   3.3%
Choose Light to Sample                            4 launches      7.59 ms /   6.3%
Sample direct - DiffuseBxDF                       4 launches      9.85 ms /   8.2%
Sample direct - CoatedDiffuseBxDF                 4 launches      2.83 ms /   2.4%
Path tracing shadow rays                          4 launches      6.84 ms /   5.7%
Sample indirect - DiffuseBxDF                     4 launches      7.36 ms /   6.1%
Sample indirect - CoatedDiffuseBxDF               4 launches      2.07 ms /   1.7%
Sample indirect - DielectricInterfaceBxDF         4 launches      0.69 ms /   0.6%
Update Film                                       2 launches      1.98 ms /   1.6%
Other                                            86 launches      1.83 ms /   1.5%
```

Performance Breakdown: San Miguel @ 1080p, 1spp

```
Reset sampler dimension                            2 launches      3.72 ms /   3.1%
Generate Camera rays                               2 launches      9.72 ms /   8.1%
Initialize PathState                               2 launches      2.71 ms /   2.3%
Clear intersections                                6 launches      5.82 ms /   4.8%
Path tracing closest hit rays                      6 launches     32.97 ms /  27.4%
Handle ray-found emission                          6 launches      2.65 ms /   2.2%
Bump and Material::GetBSDF/GetBSSRDF               4 launches     17.60 ms /  14.6%
Bump and Material::GetBSDF/GetBSSRDF               4 launches      3.98 ms /   3.3%
Choose Light to Sample                             4 launches      7.59 ms /   6.3%
Sample direct - DiffuseBxDF                        4 launches      9.85 ms /   8.2%
Sample direct - CoatedDiffuseBxDF                  4 launches      2.83 ms /   2.4%
Path tracing shadow rays                           4 launches      6.84 ms /   5.7%
Sample indirect - DiffuseBxDF                      4 launches      7.36 ms /   6.1%
Sample indirect - CoatedDiffuseBxDF                4 launches      2.07 ms /   1.7%
Sample indirect - DielectricInterfaceBxDF          4 launches      0.69 ms /   0.6%
Update Film                                        2 launches      1.98 ms /   1.6%
Other                                             86 launches      1.83 ms /   1.5%
```

Performance Breakdown:
San Miguel @ 1080p, 1spp

```
Reset sampler dimension                            2 launches       3.72 ms /   3.1%
Generate Camera rays                               2 launches       9.72 ms /   8.1%
Initialize PathState                               2 launches       2.71 ms /   2.3%
Clear intersections                                6 launches       5.82 ms /   4.8%
Path tracing closest hit rays                      6 launches      32.97 ms /  27.4%
Handle ray-found emission                          6 launches       2.65 ms /   2.2%
Bump and Material::GetBSDF/GetBSSRDF               4 launches      17.60 ms /  14.6%
Bump and Material::GetBSDF/GetBSSRDF               4 launches       3.98 ms /   3.3%
Choose Light to Sample                             4 launches       7.59 ms /   6.3%
Sample direct - DiffuseBxDF                        4 launches       9.85 ms /   8.2%
Sample direct - CoatedDiffuseBxDF                  4 launches       2.83 ms /   2.4%
Path tracing shadow rays                           4 launches       6.84 ms /   5.7%
Sample indirect - DiffuseBxDF                      4 launches       7.36 ms /   6.1%
Sample indirect - CoatedDiffuseBxDF                4 launches       2.07 ms /   1.7%
Sample indirect - DielectricInterfaceBxDF          4 launches       0.69 ms /   0.6%
Update Film                                        2 launches       1.98 ms /   1.6%
Other                                             86 launches       1.83 ms /   1.5%
```

Code Complexity

- pbrt is ~72k LOC (excluding tests, Sobol' / blue noise tables, etc.)
  - 7k LOC CPU-specific (accel structures, integrators): ~10%
  - 4k LOC GPU-specific(*) (infrastructure + path tracer, OptiX interop): ~6%
  - Shared (lights, BSDFs, materials, sampling code, …): ~84%

(*) Plus diffused impact of Allocator and tag-based dispatch

pbrt-v4 Release Plans

- SIGGRAPH: beta source code available on github
- Late 2020: online book
- Spring 2021: printed book

Summary

- GPU ray tracing is fast!
  - …even with non-ninja optimized code
- C++ was the only option for a legacy code base that still has to run on CPU; it's not necessarily the end-all GPU programming model
  - Idiomatic C++ is not necessarily optimal on the GPU..
- Programming model model design tension:
  does it all vs. provides mechanisms that let you do it all

Thanks!

- Steve Parker, Frank Jargstorf
- David Luebke, Aaron Lefohn
- James Bigler, Detlef Roettger, Keith Morley, David Hart, Ingo Wald
- Tim Foley