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
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, ...
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;
}
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;
}
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...”
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 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

• 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
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;
};
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;
        
        ...}

    std::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

```cpp
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)
```
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

Handle Emission at Intersection

Sample Medium

Evaluate Material (Simple)

Sample Light BVH

Evaluate Material (Complex)

Sample Light

Intersect Shadow

Sample BSDF

Sample Subsurface

OptiX

OptiX
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

<table>
<thead>
<tr>
<th>Task</th>
<th>Launches</th>
<th>Time (ms)</th>
<th>% of Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset sampler dimension</td>
<td>2</td>
<td>3.72</td>
<td>3.1%</td>
</tr>
<tr>
<td>Generate Camera rays</td>
<td>2</td>
<td>9.72</td>
<td>8.1%</td>
</tr>
<tr>
<td>Initialize PathState</td>
<td>2</td>
<td>2.71</td>
<td>2.3%</td>
</tr>
<tr>
<td>Clear intersections</td>
<td>6</td>
<td>5.82</td>
<td>4.8%</td>
</tr>
<tr>
<td>Path tracing closest hit rays</td>
<td>6</td>
<td>32.97</td>
<td>27.4%</td>
</tr>
<tr>
<td>Handle ray-found emission</td>
<td>6</td>
<td>2.65</td>
<td>2.2%</td>
</tr>
<tr>
<td>Bump and Material::GetBSDF/GetBSSRDF</td>
<td>4</td>
<td>17.60</td>
<td>14.6%</td>
</tr>
<tr>
<td>Bump and Material::GetBSDF/GetBSSRDF</td>
<td>4</td>
<td>3.98</td>
<td>3.3%</td>
</tr>
<tr>
<td>Choose Light to Sample</td>
<td>4</td>
<td>7.59</td>
<td>6.3%</td>
</tr>
<tr>
<td>Sample direct - DiffuseBxDF</td>
<td>4</td>
<td>9.85</td>
<td>8.2%</td>
</tr>
<tr>
<td>Sample direct - CoatedDiffuseBxDF</td>
<td>4</td>
<td>2.83</td>
<td>2.4%</td>
</tr>
<tr>
<td>Path tracing shadow rays</td>
<td>4</td>
<td>6.84</td>
<td>5.7%</td>
</tr>
<tr>
<td>Sample indirect - DiffuseBxDF</td>
<td>4</td>
<td>7.36</td>
<td>6.1%</td>
</tr>
<tr>
<td>Sample indirect - CoatedDiffuseBxDF</td>
<td>4</td>
<td>2.07</td>
<td>1.7%</td>
</tr>
<tr>
<td>Sample indirect - DielectricInterfaceBxDF</td>
<td>4</td>
<td>0.69</td>
<td>0.6%</td>
</tr>
<tr>
<td>Update Film</td>
<td>2</td>
<td>1.98</td>
<td>1.6%</td>
</tr>
<tr>
<td>Other</td>
<td>86</td>
<td>1.83</td>
<td>1.5%</td>
</tr>
</tbody>
</table>
Performance Breakdown:
San Miguel @ 1080p, 1spp

<table>
<thead>
<tr>
<th>Task</th>
<th>Launches</th>
<th>Time</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset sampler dimension</td>
<td>2</td>
<td>3.72 ms</td>
<td>3.1%</td>
</tr>
<tr>
<td>Generate Camera rays</td>
<td>2</td>
<td>9.72 ms</td>
<td>8.1%</td>
</tr>
<tr>
<td>Initialize PathState</td>
<td>2</td>
<td>2.71 ms</td>
<td>2.3%</td>
</tr>
<tr>
<td>Clear intersections</td>
<td>6</td>
<td>5.82 ms</td>
<td>4.8%</td>
</tr>
<tr>
<td>Path tracing closest hit rays</td>
<td>6</td>
<td>32.97 ms</td>
<td>27.4%</td>
</tr>
<tr>
<td>Handle ray-found emission</td>
<td>6</td>
<td>2.65 ms</td>
<td>2.2%</td>
</tr>
<tr>
<td>Bump and Material::GetBSDF/GetBSSRDF</td>
<td>4</td>
<td>17.60 ms</td>
<td>14.6%</td>
</tr>
<tr>
<td>Bump and Material::GetBSDF/GetBSSRDF</td>
<td>4</td>
<td>3.98 ms</td>
<td>3.3%</td>
</tr>
<tr>
<td>Choose Light to Sample</td>
<td>4</td>
<td>7.59 ms</td>
<td>6.3%</td>
</tr>
<tr>
<td>Sample direct - DiffuseBxDF</td>
<td>4</td>
<td>9.85 ms</td>
<td>8.2%</td>
</tr>
<tr>
<td>Sample direct - CoatedDiffuseBxDF</td>
<td>4</td>
<td>2.83 ms</td>
<td>2.4%</td>
</tr>
<tr>
<td>Path tracing shadow rays</td>
<td>4</td>
<td>6.84 ms</td>
<td>5.7%</td>
</tr>
<tr>
<td>Sample indirect - DiffuseBxDF</td>
<td>4</td>
<td>7.36 ms</td>
<td>6.1%</td>
</tr>
<tr>
<td>Sample indirect - CoatedDiffuseBxDF</td>
<td>4</td>
<td>2.07 ms</td>
<td>1.7%</td>
</tr>
<tr>
<td>Sample indirect - DielectricInterfaceBxDF</td>
<td>4</td>
<td>0.69 ms</td>
<td>0.6%</td>
</tr>
<tr>
<td>Update Film</td>
<td>2</td>
<td>1.98 ms</td>
<td>1.6%</td>
</tr>
<tr>
<td>Other</td>
<td>86</td>
<td>1.83 ms</td>
<td>1.5%</td>
</tr>
</tbody>
</table>
### Performance Breakdown:
**San Miguel @ 1080p, 1spp**

<table>
<thead>
<tr>
<th>Task</th>
<th>Launches</th>
<th>Time (ms)</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset sampler dimension</td>
<td>2</td>
<td>3.72</td>
<td>3.1%</td>
</tr>
<tr>
<td>Generate Camera rays</td>
<td>2</td>
<td>9.72</td>
<td>8.1%</td>
</tr>
<tr>
<td>Initialize PathState</td>
<td>2</td>
<td>2.71</td>
<td>2.3%</td>
</tr>
<tr>
<td>Clear intersections</td>
<td>6</td>
<td>5.82</td>
<td>4.8%</td>
</tr>
<tr>
<td>Path tracing closest hit rays</td>
<td>6</td>
<td>32.97</td>
<td>27.4%</td>
</tr>
<tr>
<td>Handle ray-found emission</td>
<td>6</td>
<td>2.65</td>
<td>2.2%</td>
</tr>
<tr>
<td>Bump and Material::GetBSDF/GetBSSRDF</td>
<td>4</td>
<td>17.60</td>
<td>14.6%</td>
</tr>
<tr>
<td>Bump and Material::GetBSDF/GetBSSRDF</td>
<td>4</td>
<td>3.98</td>
<td>3.3%</td>
</tr>
<tr>
<td>Choose Light to Sample</td>
<td>4</td>
<td>7.59</td>
<td>6.3%</td>
</tr>
<tr>
<td>Sample direct - DiffuseBxDF</td>
<td>4</td>
<td>9.85</td>
<td>8.2%</td>
</tr>
<tr>
<td>Sample direct - CoatedDiffuseBxDF</td>
<td>4</td>
<td>2.83</td>
<td>2.4%</td>
</tr>
<tr>
<td>Path tracing shadow rays</td>
<td>4</td>
<td>6.84</td>
<td>5.7%</td>
</tr>
<tr>
<td>Sample indirect - DiffuseBxDF</td>
<td>4</td>
<td>7.36</td>
<td>6.1%</td>
</tr>
<tr>
<td>Sample indirect - CoatedDiffuseBxDF</td>
<td>4</td>
<td>2.07</td>
<td>1.7%</td>
</tr>
<tr>
<td>Sample indirect - DielectricInterfaceBxDF</td>
<td>4</td>
<td>0.69</td>
<td>0.6%</td>
</tr>
<tr>
<td>Update Film</td>
<td>2</td>
<td>1.98</td>
<td>1.6%</td>
</tr>
<tr>
<td>Other</td>
<td>86</td>
<td>1.83</td>
<td>1.5%</td>
</tr>
</tbody>
</table>
Performance Breakdown:
San Miguel @ 1080p, 1spp

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

**San Miguel @ 1080p, 1spp**

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