Developing V-Ray Next

Vladimir Koylazov

Total Chaos 2018

Overview

- Three stories of heroism, despair and (mostly) happy endings:
  - Wrapping up the Embree integration and update to Embree 2.13;
  - SIMD-ifying the V-Ray code;
  - Adaptive dome light challenges.

Embree integration

- Originally we had our own intersection code:
  - Static (non-motion-blurred geometry)
  - Motion-blurred geometry
  - Instances/proxies
  - Hair
  - Particles
  - Displacement
- Embree was introduced gradually for different types of intersections
  - Started with V-Ray 3.0 for static and single-segment motion blur
  - Hair, instances, proxies added over the various 3.x releases
- Embree is generally 2x faster than our intersectors
  - Translates to ~25% lower render times

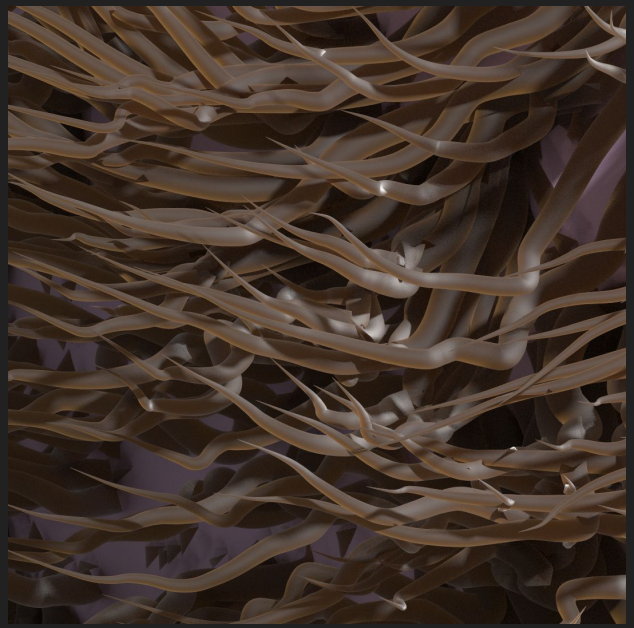
Embree integration in V-Ray Next

- Updated to Embree 2.13
  - Mostly because of some additional hair features
  - Quad intersections seemed something worth exploring
- Added multisegmented motion blur support
- Will work on reducing memory usage over the V-Ray Next dev cycle

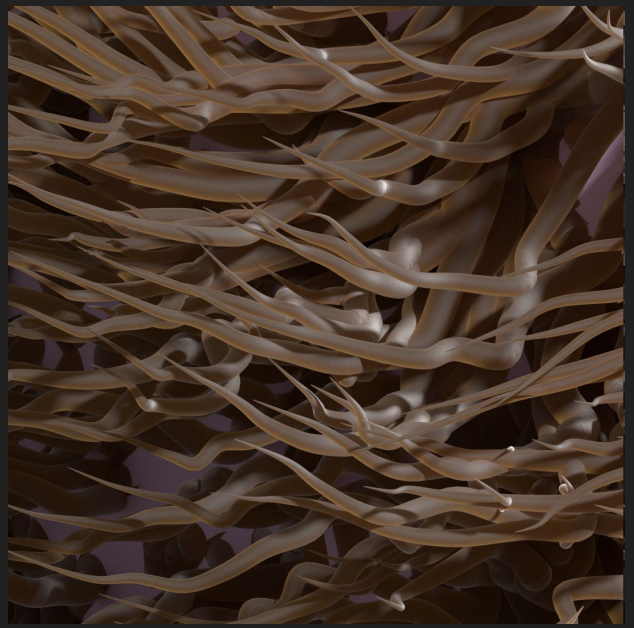
Custom Embree modifications

- We have a few custom Embree modifications
  - Skip tags;
  - Fat hair intersectors;
  - Custom geometry layout for conserving memory
- We had to keep our changes separate from the main Embree code

Embree fat hair intersector

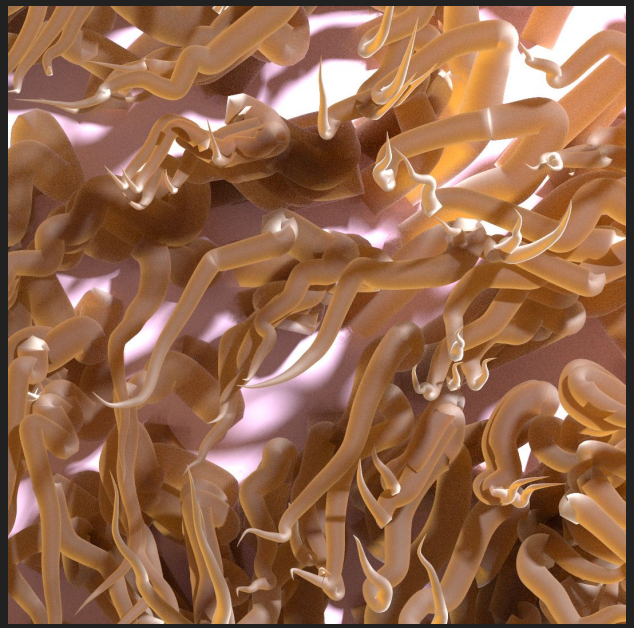


Original Embree
hair intersector

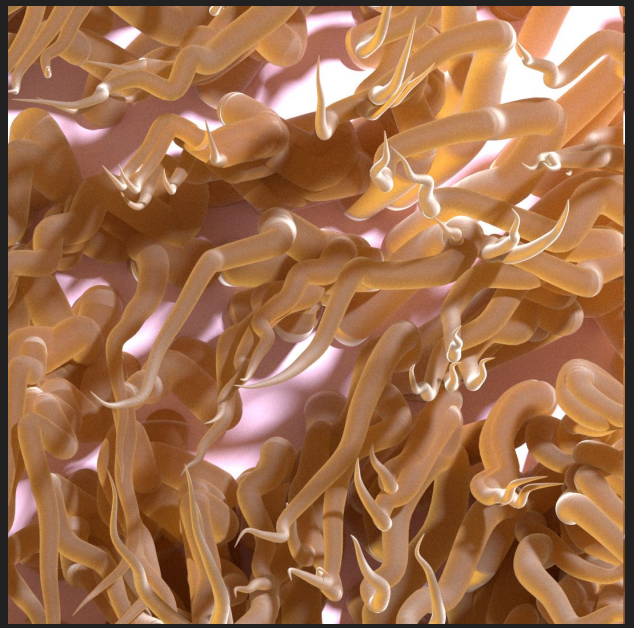


Our fat hair
intersector

Embree fat hair intersector



Original intersector



Fat intersector

Embree 2.13 - porting our custom changes

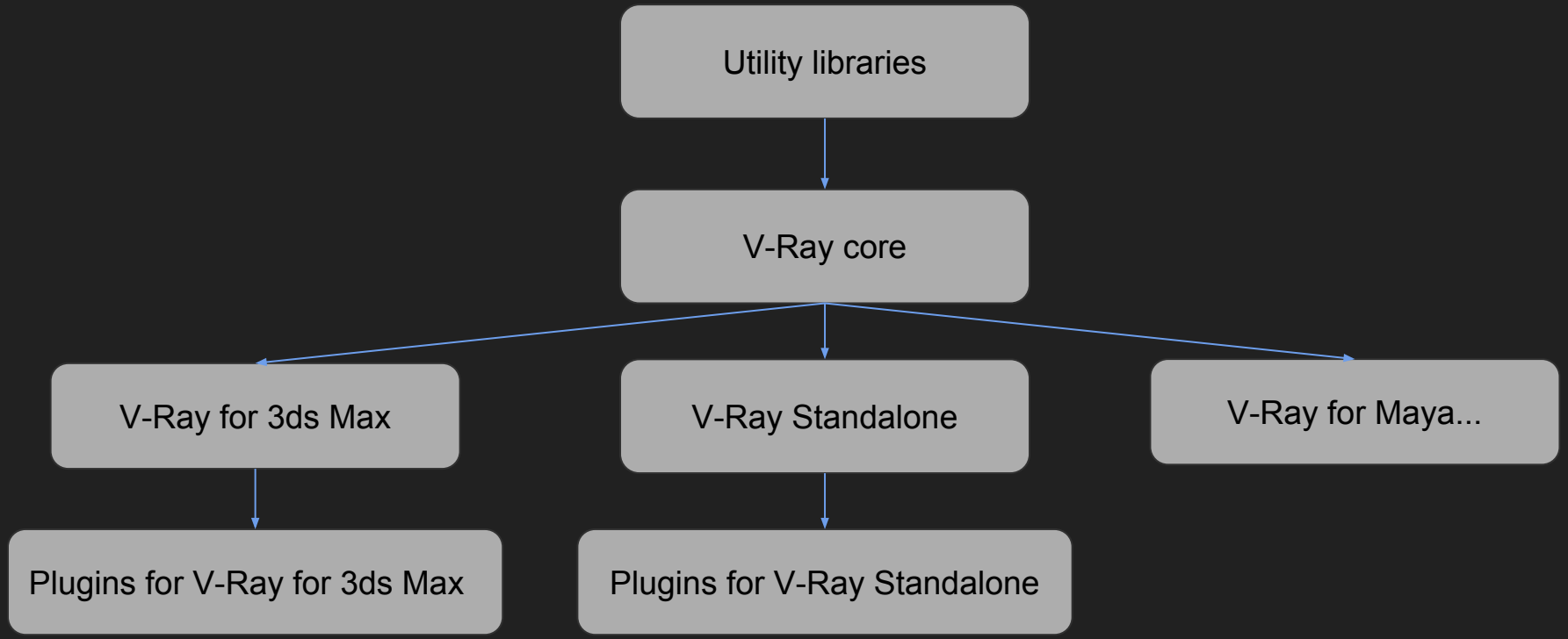
- When we compared the code for Embree 2.3 and Embree 2.13…
  - …the code was totally different and refactored;
  - It was not immediately clear how to port our modifications to the new Embree code base
- Martin Krastev spent a whole year rebasing our changes on top of the Embree 2.13 code…
  - ...commit by commit
  - Fixing any problems along the way.
- Unit tests were extremely important

Embree 2.13

SIMD-ifying the V-Ray code

- SSE2 instructions
  - Provide efficient implementation of 4-component vector, color and matrix operations
  - In limited tests showed measurable performance improvements
- Not available in 2000 when the V-Ray math library was first developed
- Vectors, colors and matrices are used throughout the V-Ray code
  - Some of the code used double-precision calculations as well which are (much) slower
- How can this change be made?
  - In a reasonable time frame;
  - Without breaking anything.
- Staged implementation:
  - First for the intersection code of rays with geometry primitives;
  - Then for some vector shading elements (surface normals, intersection points);
  - Finally, for colors and all members of the ray state

V-Ray code basic structure



SIMD-ifying the V-Ray code

- The goal was clear…
  - ...but where to start?
- There are thousands of source files and vectors/colors/matrices are used in thousands of places
  - ~3500 .cpp files, ~5500 .h and .hpp files
  - Together vector types occur on 46679 lines in 2787 files
- Not all occurrences need to be changed
  - Storage of vector/color data in large arrays in memory needs to remain as non-SIMD types
    - For RAM usage reasons
    - For alignment reasons
  - Some calculations are not performance-critical

“Show me your [code] and conceal your [data structures], and I shall continue to be mystified. Show me your [data structures], and I won't usually need your [code]; it'll be obvious.”

- *After Fred Brooks,*
  “The Mythical Man-Month”

SIMD-ifying the V-Ray code

- Start by modifying the key data structures used in the code
  - Class Ray and TraceRay
  - Class IntersectionData
  - Class VRayContext
- First do vector math, then do colors
- Make sure that SIMD classes have the same interface as non-SIMD classes
  - So that in most cases the change would be simply replacing the type of a variable/function argument

Edit

vray/vray/include/vrayinterface.h: 70a04ba

```
923 /// Describes an intersection of a ray with a surface
924 struct IntersectionData {
925     /// The primitive that produced the intersection
926     GenericPrimitive *primitive;
927
928     /// A pointer to a shadeable object (makes no sense to be NULL, but may be)
929     Shadeable *sb;
930
931     /// A pointer to additional shading data
932     VRayShadeInstance *si;
933
934     /// A pointer to additional texture-mapping data
935     VRayShadeData *sd;
936
937     /// A pointer to a volume shader for the object
938     VRayVolume *volume;
939
940     /// A pointer to additional surface properties
941     VRaySurfaceProperties *surfaceProps;
942
943     void *skipTag; ///< A single render primitive to exclude while tracing the ray (for avoiding "surface acne"), may be
944
945     Ireal wpointCoeff; ///< The distance along the ray where the intersection occurred
946
947     /// The intersection point itself in internal space. In versions of the V-Ray SDK prior to 1.90.00, this
948     /// point was directly in world space. In versions 1.90.00 and later, the actual world-space coordinates
949     /// can be obtained by adding VRayFrameData::sceneOffset to the wpoint.
950     TracePoint wpoint;
951
952     Vector normal; ///< The smooth normal at the surface point. In world coordinates, may be non-unit when passed to VRa
953     Vector gnormal; ///< The geometric normal at the surface point. In world coordinates, may be non-unit when passed to
954
955     // The following members are set and used by VRay's shadeable objects to pass information to the ShadeData object
956     // You can use them for whatever reasons are necessary
957
958     int faceIndex; ///< Index of the intersected primitive, used to identify it to the Renderable and also for material
959     Vector bary; ///< Barycentric coordinates of the intersection; handy for triangle meshes
960     Vector faceBase, faceEdge0, faceEdge1; ///< The intersected "face" in world coordinates, for texture filtering and b
961
962     // Some additional stuff for CA Scanline, to be removed later
963     Color atmosColor;
964     Color atmosTransp;
965
966     /// Additional stuff if needed
967     union {
968         int extra0;
969         float extraf;
970         void *extrap;
971         int extra_int[2];
972     };
973
974     /// Clears the IntersectionData to repsent an empty intersection
975     void clear(void) {
976         primitive=NULL;
977         sb=NULL;
978         si=NULL;
979         sd=NULL;
980         volume=NULL;
981         surfaceProps=NULL;
982         skipTag=NULL;
983     }
984
985     /// @name Accessor methods - provide better compatibility between versions
986     /// @{
987     VRayExport GenericPrimitive* getPrimitive(void) const;
988     VRayExport void setPrimitive(GenericPrimitive* primitive);
989
990     VRayExport Shadeable* getShadeable(void) const;
991     VRayExport void setShadeable(Shadeable *shadeable);
992
```

vray/vray/include/vrayinterface.h: 73dea99

```
946 /// Describes an intersection of a ray with a surface
947 struct IntersectionData {
948     /// The primitive that produced the intersection
949     GenericPrimitive *primitive;
950
951     /// A pointer to a shadeable object (makes no sense to be NULL, but may be)
952     Shadeable *sb;
953
954     /// A pointer to additional shading data
955     VRayShadeInstance *si;
956
957     /// A pointer to additional texture-mapping data
958     VRayShadeData *sd;
959
960     /// A pointer to a volume shader for the object
961     VRayVolume *volume;
962
963     /// A pointer to additional surface properties
964     VRaySurfaceProperties *surfaceProps;
965
966     void *skipTag; ///< A single render primitive to exclude while tracing the ray (for avoiding "surface acne"), may be
967
968     Ireal wpointCoeff; ///< The distance along the ray where the intersection occurred
969
970     /// The intersection point itself in internal space. In versions of the V-Ray SDK prior to 1.90.00, this
971     /// point was directly in world space. In versions 1.90.00 and later, the actual world-space coordinates
972     /// can be obtained by adding VRayFrameData::sceneOffset to the wpoint.
973     ShadeVec wpoint;
974
975     ShadeVec normal; ///< The smooth normal at the surface point. In world coordinates, may be non-unit when passed to VR
976     ShadeVec gnormal; ///< The geometric normal at the surface point. In world coordinates, may be non-unit when passed
977
978     // The following members are set and used by VRay's shadeable objects to pass information to the ShadeData object
979     // You can use them for whatever reasons are necessary
980
981     int faceIndex; ///< Index of the intersected primitive, used to identify it to the Renderable and also for material
982     ShadeVec bary; ///< Barycentric coordinates of the intersection; handy for triangle meshes
983     ShadeVec faceBase, faceEdge0, faceEdge1; ///< The intersected "face" in world coordinates, for texture filtering and
984
985     // Some additional stuff for CA Scanline, to be removed later
986     Color atmosColor;
987     Color atmosTransp;
988
989     /// Additional stuff if needed
990     union {
991         int extra0;
992         float extraf;
993         void *extrap;
994         int extra_int[2];
995     };
996
997     /// Clears the IntersectionData to repsent an empty intersection
998     void clear(void) {
999         primitive=NULL;
1000         sb=NULL;
1001         si=NULL;
1002         sd=NULL;
1003         volume=NULL;
1004         surfaceProps=NULL;
1005         skipTag=NULL;
1006     }
1007
1008     /// @name Accessor methods - provide better compatibility between versions
1009     /// @{
1010     VRayExport GenericPrimitive* getPrimitive(void) const;
1011     VRayExport void setPrimitive(GenericPrimitive* primitive);
1012
1013     VRayExport Shadeable* getShadeable(void) const;
1014     VRayExport void setShadeable(Shadeable *shadeable);
1015
```

SIMD-ifying the V-Ray code

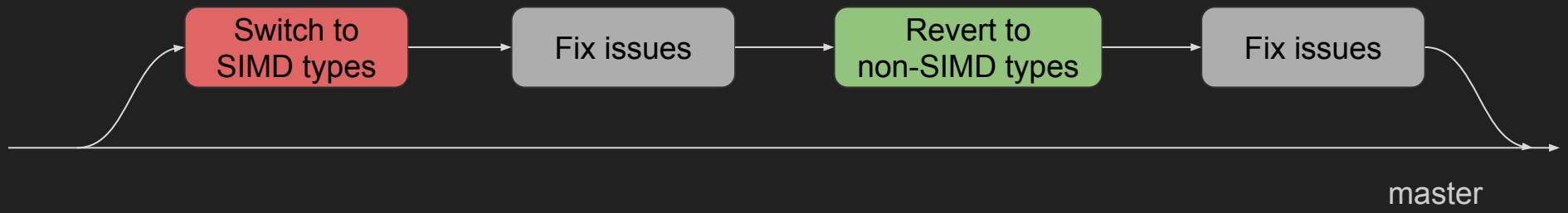
- Initial idea for each stage: work in a branch
  - Start by modifying the base raycasting classes (i.e. class Ray, TraceRay, IntersectionData);
  - Make sure we get compiler errors between incompatible classes;
    - Convert explicitly from/between non-SIMD classes as needed;
    - Try to keep all intermediate calculations SIMD-ified;
  - Resolve all compiler errors;
  - Merge the branch into the master.
- Problem:
  - The master branch changes daily;
  - Keeping the branch in sync with the master is exhausting.
  - Some projects (like Phoenix FD) need to compile both against V-Ray 3.x and V-Ray Next SDK.

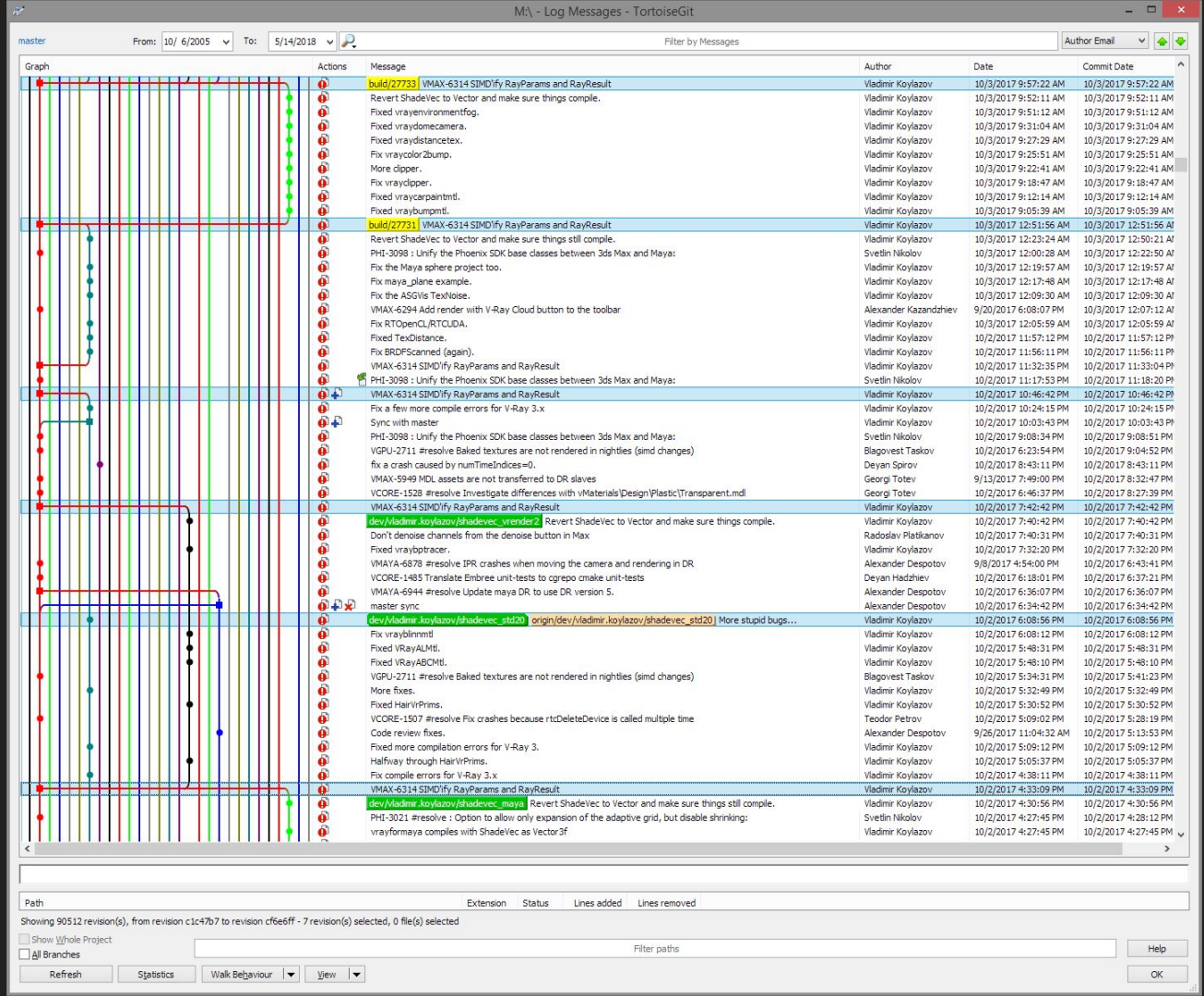
SIMD-ifying the V-Ray code

- Solution:
  - Define a new type that can map to either a non-SIMD type, or a SIMD type with an #ifdef
  - Provide explicit conversion functions between types (no implicit type conversions!)
  - Work in short branches:
    - Change the #ifdef to the SIMD type
    - Fix all compilation errors in a given portion of the code (a library, or a bunch of .cpp files)
    - Change the #ifdef back to the non-SIMD type
    - Make sure things still compile
    - Merge into the master
    - Repeat until an executable product builds with the SIMD version so that it can be tested
    - Complete the changes for the rest of the products
  - The final switch is just changing a define

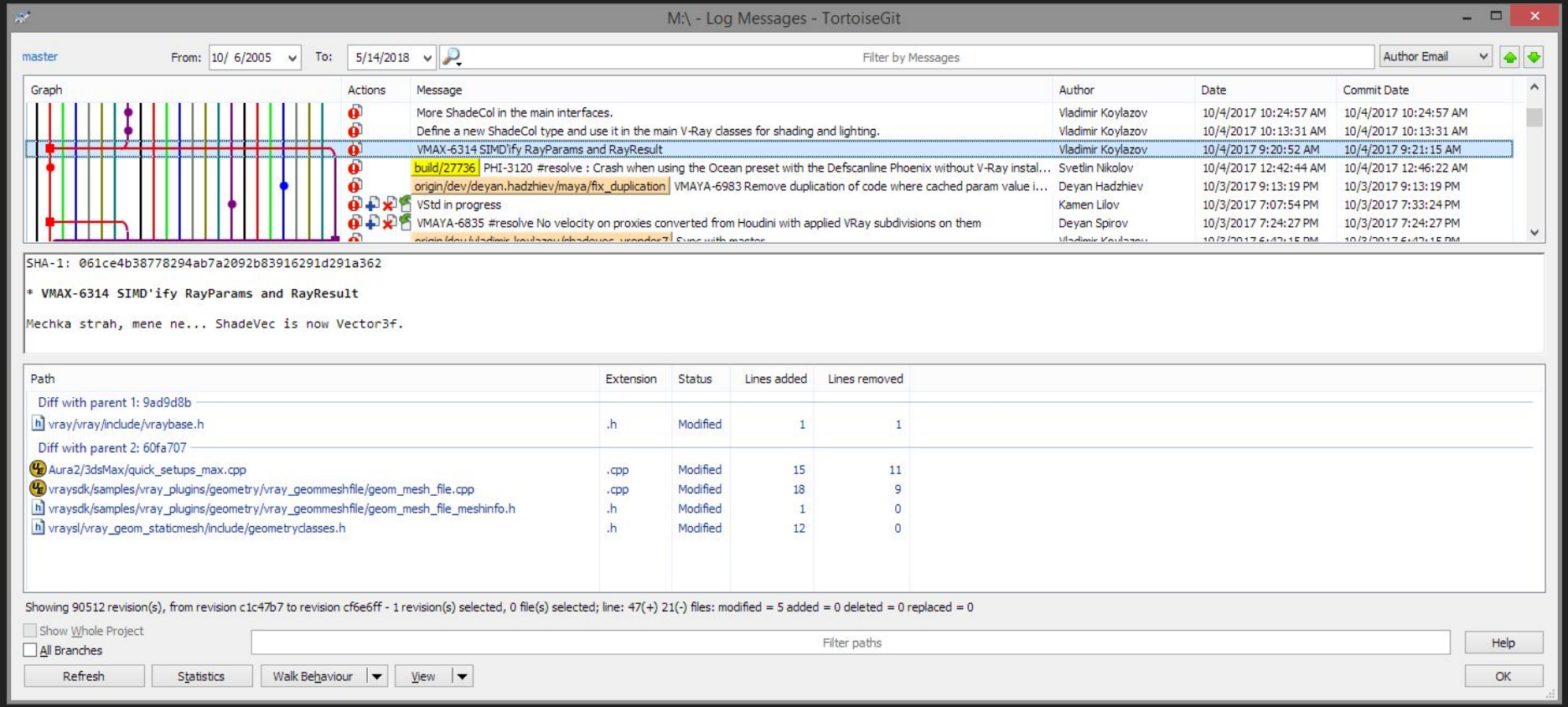
```
101
102 // Define the vector type to use when shading; will eventually be replaced
103 // with just one or the other once everything is converted
104 #if 1
105
106 typedef simd::Vector3f ShadeVec;
107 typedef simd::Transform3x4f ShadeTransform;
108 typedef simd::Matrix3x3f ShadeMatrix;
109
110 FORCEINLINE ShadeVec toShadeVec(const Vector &v) { return simd::Vector3f(v); }
111 FORCEINLINE ShadeVec toShadeVec(const simd::Vector3f &v) { return v; }
112 FORCEINLINE ShadeVec toShadeVec(const TracePoint &v) { return simd::Vector3f(v.x, v.y, v.z); }
113
114 FORCEINLINE ShadeMatrix toShadeMatrix(const Matrix &m) { return simd::Matrix3x3f(m); }
115 FORCEINLINE ShadeMatrix toShadeMatrix(const simd::Matrix3x3f &m) { return m; }
116
117 FORCEINLINE ShadeTransform toShadeTransform(const Transform &m) { return simd::Transform3x4f(m); }
118 FORCEINLINE ShadeTransform toShadeTransform(const simd::Transform3x4f &m) { return m; }
119 FORCEINLINE ShadeTransform toShadeTransform(const TraceTransform &m) {
120     simd::Transform3x4f res;
121     res.m[0].set(m.m[0]);
122     res.m[1].set(m.m[1]);
123     res.m[2].set(m.m[2]);
124     res.offs.set((float) m.offs.x, (float) m.offs.y, (float) m.offs.z);
125     return res;
126 }
127
128 FORCEINLINE Vector toVector(const ShadeVec &v) { return v.toVector(); }
129 FORCEINLINE simd::Vector3f toVector3f(const ShadeVec &v) { return v; }
130 FORCEINLINE simd::Vector3f toVector3f(const TracePoint &v) { return simd::Vector3f(v.x, v.y, v.z); }
131 FORCEINLINE Color toColor(const ShadeVec &v) { return Color(v.x(), v.y(), v.z()); }
132 FORCEINLINE Transform toTransform(const ShadeTransform &tm) { return tm.toTransform(); }
133 FORCEINLINE TraceTransform toTraceTransform(const ShadeTransform &tm) {
134     TraceTransform res;
135     res.m=tm.m.toMatrix();
136     res.offs=TracePoint(tm.offs.x(), tm.offs.y(), tm.offs.z());
137     return res;
138 }
139
140 FORCEINLINE Matrix toMatrix(const ShadeMatrix &m) { return m.toMatrix(); }
141
142 #define SHADEVEC_IS_VECTOR3F
143
144 #else
145
146 typedef Vector ShadeVec;
147 typedef Transform ShadeTransform;
148 typedef Matrix ShadeMatrix;
149
150 FORCEINLINE ShadeVec toShadeVec(const Vector &v) { return v; }
151 FORCEINLINE ShadeVec toShadeVec(const simd::Vector3f &v) { return v.toVector(); }
152 FORCEINLINE ShadeVec toShadeVec(const TracePoint &v) { return v.toVector(); }
153
154 FORCEINLINE ShadeMatrix toShadeMatrix(const Matrix &m) { return m; }
155 FORCEINLINE ShadeMatrix toShadeMatrix(const simd::Matrix3x3f &m) { return Matrix(m[0].toVector(), m[1].toVector(), m[2].toVector()); }
156
157 FORCEINLINE ShadeTransform toShadeTransform(const Transform &m) { return m; }
158 FORCEINLINE ShadeTransform toShadeTransform(const simd::Transform3x4f &m) { return Transform(Matrix(m.m[0].toVector(), m.m[1].toVector(), m.m[2].toVector()), m.offs.toVector()); }
159 FORCEINLINE ShadeTransform toShadeTransform(const TraceTransform &m) {
160     Transform res;
161     res.m[0]=m.m[0];
162     res.m[1]=m.m[1];
163     res.m[2]=m.m[2];
164     res.offs=m.offs.toVector();
165     return res;
166 }
167
168 FORCEINLINE Vector toVector(const ShadeVec &v) { return v; }
169 FORCEINLINE simd::Vector3f toVector3f(const ShadeVec &v) { return simd::Vector3f(v); }
170 FORCEINLINE simd::Vector3f toVector3f(const TracePoint &v) { return simd::Vector3f(v.x, v.y, v.z); }
171 FORCEINLINE Color toColor(const ShadeVec &v) { return Color(v.x, v.y, v.z); }
172 FORCEINLINE Transform toTransform(const ShadeTransform &tm) { return tm; }
173 FORCEINLINE TraceTransform toTraceTransform(const ShadeTransform &tm) { return TraceTransform(tm); }
174 FORCEINLINE Matrix toMatrix(const ShadeMatrix &m) { return m; }
175
176 #define SHADEVEC_IS_VECTOR
177
178 #endif
```

SIMD-ifying the V-Ray code




M:\ - Log Messages - TortoiseGit
master
From: 10/ 6/2005
To: 5/14/2018
Filter by Messages
Author Email
Graph
Actions
Message
Author
Date
Commit Date
build/27733 VMAX-6314 SIMD\fy RayParams and RayResult Vladimir Koylazov 10/3/2017 9:57:22 AM 10/3/2017 9:57:22 AM
Revert ShadeVec to Vector and make sure things compile. Vladimir Koylazov 10/3/2017 9:52:11 AM 10/3/2017 9:52:11 AM
Fixed vrayenvironmentfog. Vladimir Koylazov 10/3/2017 9:51:12 AM 10/3/2017 9:51:12 AM
Fixed vraydomecamera. Vladimir Koylazov 10/3/2017 9:31:04 AM 10/3/2017 9:31:04 AM
Fixed vraydistancetex. Vladimir Koylazov 10/3/2017 9:27:29 AM 10/3/2017 9:27:29 AM
Fix vraycolor2bump. Vladimir Koylazov 10/3/2017 9:25:51 AM 10/3/2017 9:25:51 AM
More clipper. Vladimir Koylazov 10/3/2017 9:22:41 AM 10/3/2017 9:22:41 AM
Fix vrayclipper. Vladimir Koylazov 10/3/2017 9:18:47 AM 10/3/2017 9:18:47 AM
Fixed vraycarpaintmtl. Vladimir Koylazov 10/3/2017 9:12:14 AM 10/3/2017 9:12:14 AM
Fixed vraybumpmtl. Vladimir Koylazov 10/3/2017 9:05:39 AM 10/3/2017 9:05:39 AM
build/27731 VMAX-6314 SIMD\fy RayParams and RayResult Vladimir Koylazov 10/3/2017 12:51:56 AM 10/3/2017 12:51:56 AM
Revert ShadeVec to Vector and make sure things still compile. Vladimir Koylazov 10/3/2017 12:23:24 AM 10/3/2017 12:50:21 AM
PHI-3098 : Unify the Phoenix SDK base classes between 3ds Max and Maya: Svetlin Nikolov 10/3/2017 12:00:28 AM 10/3/2017 12:22:50 AM
Fix the Maya sphere project too. Vladimir Koylazov 10/3/2017 12:19:57 AM 10/3/2017 12:19:57 AM
Fix maya_plane example. Vladimir Koylazov 10/3/2017 12:17:48 AM 10/3/2017 12:17:48 AM
Fix the ASGVis TexNoise. Vladimir Koylazov 10/3/2017 12:09:30 AM 10/3/2017 12:09:30 AM
VMAX-6294 Add render with V-Ray Cloud button to the toolbar Alexander Kazandzhiev 9/20/2017 6:08:07 PM 10/3/2017 12:07:12 AM
Fix RTOpenCL/RTCUDA. Vladimir Koylazov 10/3/2017 12:05:59 AM 10/3/2017 12:05:59 AM
Fixed TexDistance. Vladimir Koylazov 10/2/2017 11:57:12 PM 10/2/2017 11:57:12 PM
Fix BRDFScanned (again). Vladimir Koylazov 10/2/2017 11:56:11 PM 10/2/2017 11:56:11 PM
VMAX-6314 SIMD\fy RayParams and RayResult Vladimir Koylazov 10/2/2017 11:32:35 PM 10/2/2017 11:33:04 PM
PHI-3098 : Unify the Phoenix SDK base classes between 3ds Max and Maya: Svetlin Nikolov 10/2/2017 11:17:53 PM 10/2/2017 11:18:20 PM
VMAX-6314 SIMD\fy RayParams and RayResult Vladimir Koylazov 10/2/2017 10:46:42 PM 10/2/2017 10:46:42 PM
Fix a few more compile errors for V-Ray 3.x Vladimir Koylazov 10/2/2017 10:24:15 PM 10/2/2017 10:24:15 PM
Sync with master Vladimir Koylazov 10/2/2017 10:03:43 PM 10/2/2017 10:03:43 PM
PHI-3098 : Unify the Phoenix SDK base classes between 3ds Max and Maya: Svetlin Nikolov 10/2/2017 9:08:34 PM 10/2/2017 9:08:51 PM
VGPU-2711 #resolve Baked textures are not rendered in nightles (simd changes) Blagovest Taskov 10/2/2017 6:23:54 PM 10/2/2017 9:04:52 PM
fix a crash caused by numTimeIndices=0. Deyan Spirov 10/2/2017 8:43:11 PM 10/2/2017 8:43:11 PM
VMAX-5949 MDL assets are not transferred to DR slaves Georgi Totev 9/13/2017 7:49:00 PM 10/2/2017 8:32:47 PM
VCORE-1528 #resolve Investigate differences with vMaterials\Design\Plastic\Transparent.mdl Georgi Totev 10/2/2017 6:46:37 PM 10/2/2017 8:27:39 PM
VMAX-6314 SIMD\fy RayParams and RayResult Vladimir Koylazov 10/2/2017 7:42:42 PM 10/2/2017 7:42:42 PM
dev/vladimir.koylazov/shadevec_vrender2 Revert ShadeVec to Vector and make sure things compile. Vladimir Koylazov 10/2/2017 7:40:42 PM 10/2/2017 7:40:42 PM
Don't denoise channels from the denoise button in Max Radoslav Platikanov 10/2/2017 7:40:31 PM 10/2/2017 7:40:31 PM
Fixed vraybptracer. Vladimir Koylazov 10/2/2017 7:32:20 PM 10/2/2017 7:32:20 PM
VMAYA-6878 #resolve IPR crashes when moving the camera and rendering in DR Alexander Despotov 9/8/2017 4:54:00 PM 10/2/2017 6:43:41 PM
VCORE-1485 Translate Embree unit-tests to cgrepo cmake unit-tests Deyan Hadzhiev 10/2/2017 6:18:01 PM 10/2/2017 6:37:21 PM
VMAYA-6944 #resolve Update maya DR to use DR version 5. Alexander Despotov 10/2/2017 6:36:07 PM 10/2/2017 6:36:07 PM
master sync Alexander Despotov 10/2/2017 6:34:42 PM 10/2/2017 6:34:42 PM
dev/vladimir.koylazov/shadevec_std20 origin/dev/vladimir.koylazov/shadevec_std20 More stupid bugs... Vladimir Koylazov 10/2/2017 6:08:56 PM 10/2/2017 6:08:56 PM
Fix vrayblinnmtl Vladimir Koylazov 10/2/2017 6:08:12 PM 10/2/2017 6:08:12 PM
Fixed VRayALMtl. Vladimir Koylazov 10/2/2017 5:48:31 PM 10/2/2017 5:48:31 PM
Fixed VRayABCMtl. Vladimir Koylazov 10/2/2017 5:48:10 PM 10/2/2017 5:48:10 PM
VGPU-2711 #resolve Baked textures are not rendered in nightles (simd changes) Blagovest Taskov 10/2/2017 5:34:31 PM 10/2/2017 5:41:23 PM
More fixes. Vladimir Koylazov 10/2/2017 5:32:49 PM 10/2/2017 5:32:49 PM
Fixed HairVrPrims. Vladimir Koylazov 10/2/2017 5:30:52 PM 10/2/2017 5:30:52 PM
VCORE-1507 #resolve Fix crashes because rtcDeleteDevice is called multiple time Teodor Petrov 10/2/2017 5:09:02 PM 10/2/2017 5:28:19 PM
Code review fixes. Alexander Despotov 9/26/2017 11:04:32 AM 10/2/2017 5:13:53 PM
Fixed more compilation errors for V-Ray 3. Vladimir Koylazov 10/2/2017 5:09:12 PM 10/2/2017 5:09:12 PM
Halfway through HairVrPrims. Vladimir Koylazov 10/2/2017 5:05:37 PM 10/2/2017 5:05:37 PM
Fix compile errors for V-Ray 3.x Vladimir Koylazov 10/2/2017 4:38:11 PM 10/2/2017 4:38:11 PM
VMAX-6314 SIMD\fy RayParams and RayResult Vladimir Koylazov 10/2/2017 4:33:09 PM 10/2/2017 4:33:09 PM
dev/vladimir.koylazov/shadevec_maya Revert ShadeVec to Vector and make sure things still compile. Vladimir Koylazov 10/2/2017 4:30:56 PM 10/2/2017 4:30:56 PM
PHI-3021 #resolve : Option to allow only expansion of the adaptive grid, but disable shrinking: Svetlin Nikolov 10/2/2017 4:27:45 PM 10/2/2017 4:28:12 PM
vrayformaya compiles with ShadeVec as Vector3f Vladimir Koylazov 10/2/2017 4:27:45 PM 10/2/2017 4:27:45 PM
Path
Extension
Status
Lines added
Lines removed
Showing 90512 revision(s), from revision c1c47b7 to revision cf6e6ff - 7 revision(s) selected, 0 file(s) selected
Show Whole Project
All Branches
Filter paths
Help
Refresh
Statistics
Walk Behaviour
View
OK

Final switch to SIMD type



SIMD-ifying the V-Ray code

- Worked surprisingly well
  - It helped a lot that the V-Ray code base is modular
  - Of course, some minor bugs did occur and they were later found and fixed during testing
- For the three stages, three new types were introduced:
  - Float3 can map to either Vector or simd::Vector3f
    - Used in intersection libraries
  - ShadeVec can map to either Vector or simd::Vector3f
    - Used in V-Ray as part of the ray context
  - ShadeCol can map to either Color or simd::Color3f
    - Used in V-Ray as part of the ray context
- We had to go through the *entire* V-Ray code base
  - Multiple times (for each stage)
  - Practically every single file was touched in some way

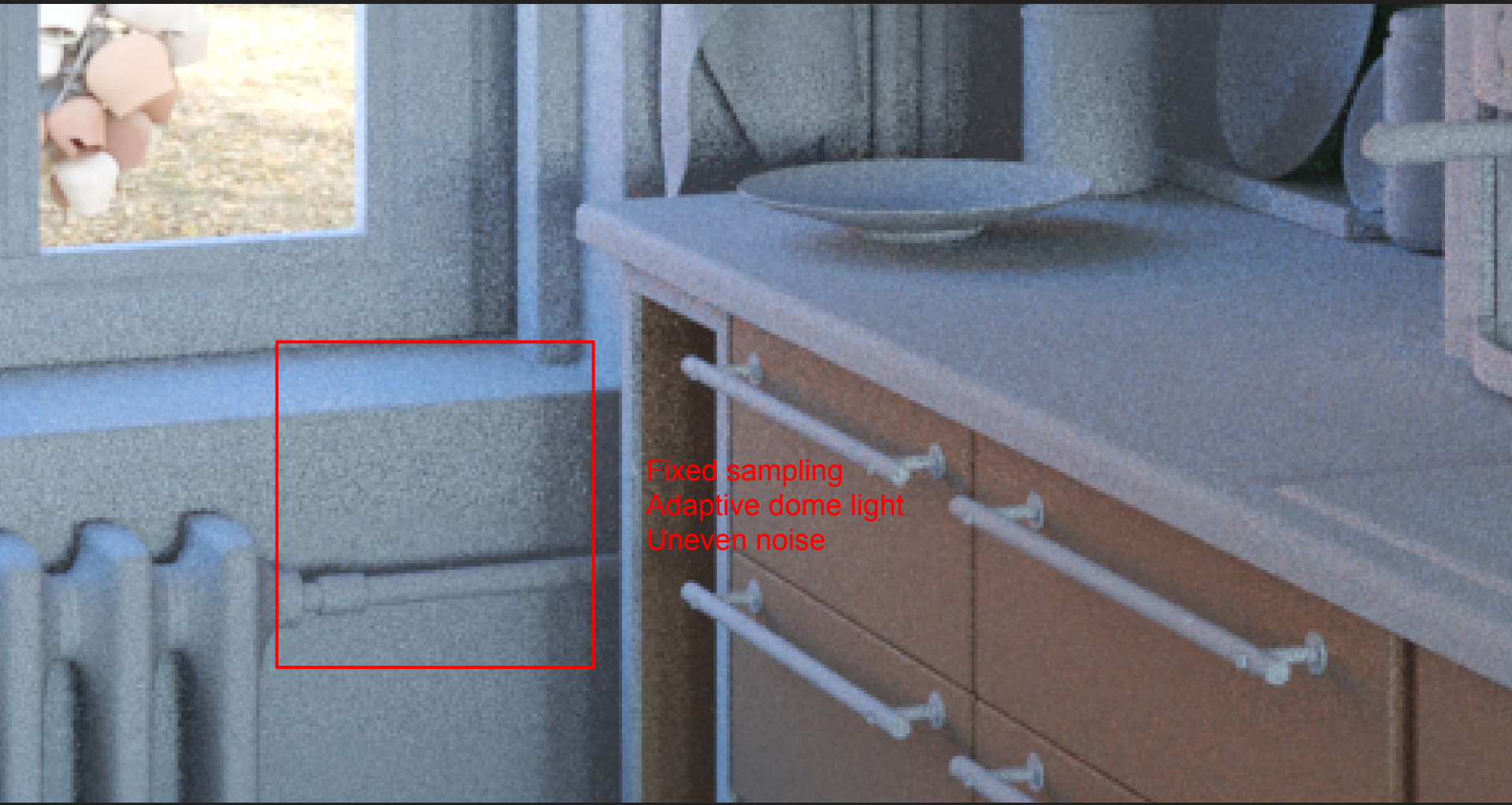
SIMD-ifying the V-Ray code

- The result
  - Up to 25% faster rendering
  - A compatibility header for compiling shaders both for V-Ray 3.x and V-Ray Next
- Some calculations were not SIMD-ified
  - Diminishing returns
- Work took about 2 months

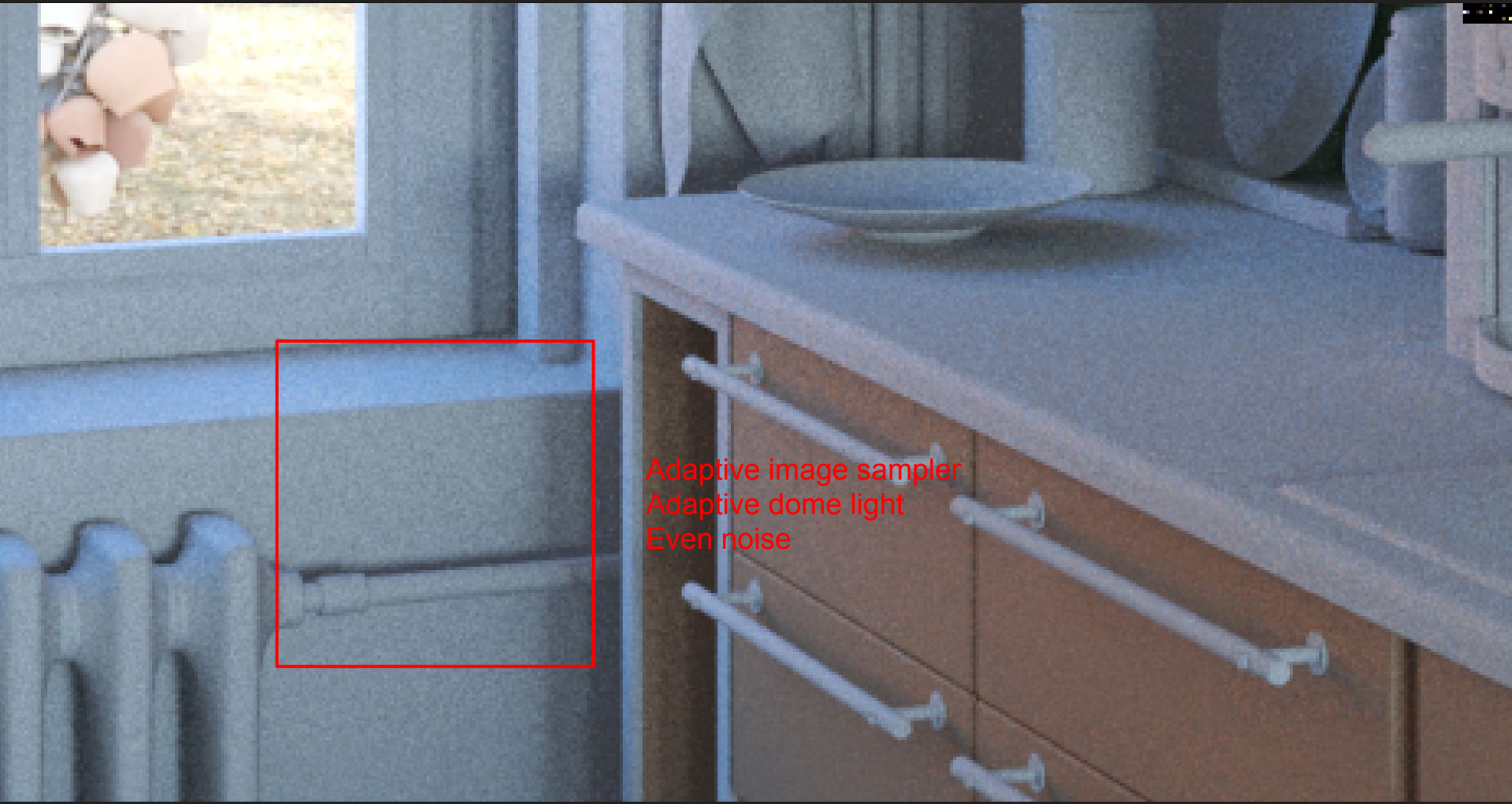
Adaptive dome light

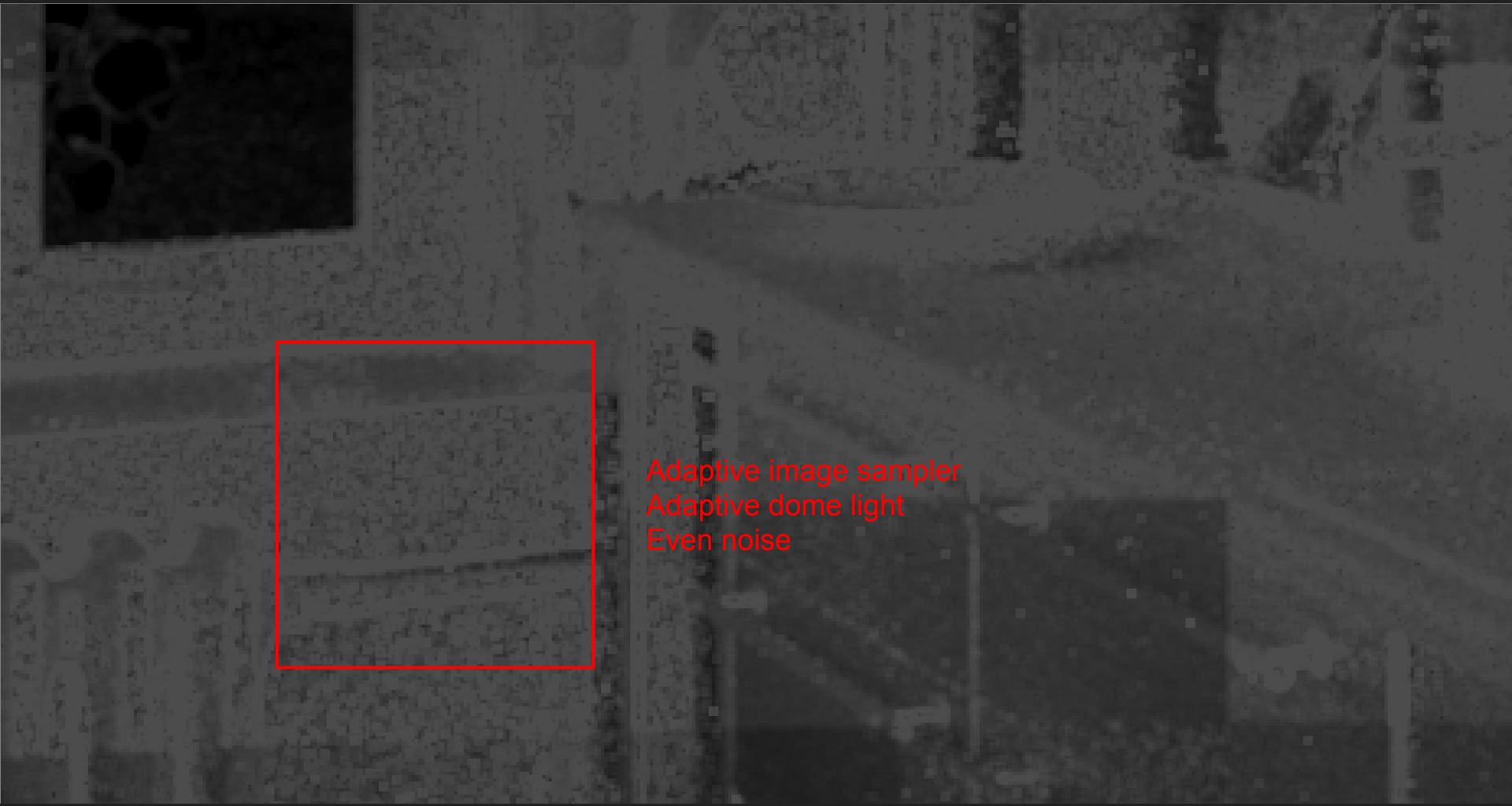
Adaptive dome light challenges

- A continuation of the work we did on the adaptive lights to make V-Ray smarter
  - Basic idea is fairly simple - use the light cache to figure out which parts of the dome light are important to which parts of the scene
  - Use this information to improve sampling during the actual rendering
  - A talk about it at Siggraph 2018
- Requires good adaptive image sampling that can handle the uneven light sampling


Fixed sampling
Adaptive dome light
Uneven noise


Fixed sampling
Adaptive dome light
Uneven noise


Adaptive image sampler
Adaptive dome light
Even noise


Adaptive image sampler
Adaptive dome light
Even noise

Adaptive dome light challenges

- Asen Atanasov did some initial experiments
  - Unfortunately the performance was not as we expected
  - After a month or so, we were ready to give up

The importance of code comments - actual text

```
215     ///·Select·a·light·source·from·a·given·random·number.
216     ///·@param[in]·x·A·random·number·in·[0,1)·used·to·select·a·light·source·from·this·cell.
217     ///·@param[out]·prob·The·probability·for·selecting·the·light·source.
218     ///·@retval·The·index·of·the·selected·light·source.
219     int·getLight(float·&x,·float·&prob)·const·{
220         const·int·numLights=lightInfos.count();
221
222         int·lightIndex;
223         //·Use·binary·search·to·locate·the·light·source·that·corresponds·to·x
224         if·(x<=lightInfos[0].getValue())·lightIndex=0;
225         else·lightIndex=Min(numLights-1,·binarySearch(x,·numLights)+1);
226
227         //·Compute·the·probability·for·selecting·the·light·source.
228         prob=getLightProb(lightIndex);
229         return·lightIndex;
230     }
```

What the comment should have said...

```
246     /// Select a light source from a given random number.
247     /// @param[in] x A random number in [0,1) used to select a light source from this cell.
248     /// @param[out] invProb The inverse probability for selecting the light source.
249     /// @retval The index of the selected light source.
250     int getLight(float &x, float &invProb) const {
251         const int numLights=lightInfos.count();
252 
253         int lightIndex;
254         // Use binary search to locate the light source that corresponds to x
255         if (x<=lightInfos[0].getValue()) lightIndex=0;
256         else lightIndex=Min(numLights-1, binarySearch(x, numLights)+1);
257 
258         // Compute the inverse probability for selecting the light source.
259         invProb=getInvLightProb(lightIndex);
260         return lightIndex;
261     }
```

Adaptive dome light

- Initial tests very promising
  - Between 1.5-7x speed improvements
- Released in beta 1, all was fine…
- ...until some users reported blocky artifacts

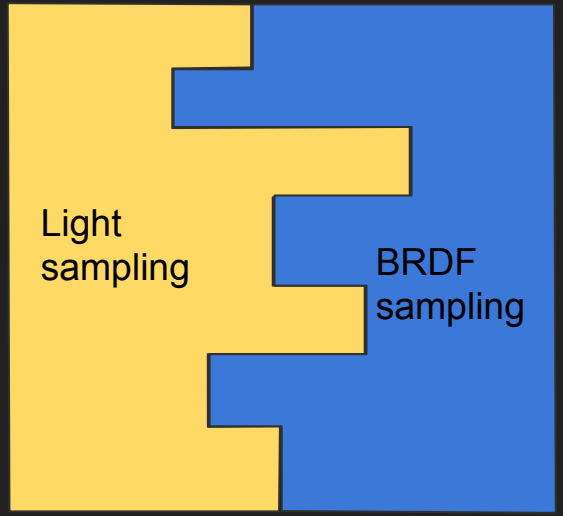
User reports for blocky artifacts

Adaptive dome light

- Debugged the user scene to see what the problem was...
- ...and after I realized what it was, I was totally terrified.
  - The whole approach could turn out to be pointless or with little practical use

Multiple importance sampling

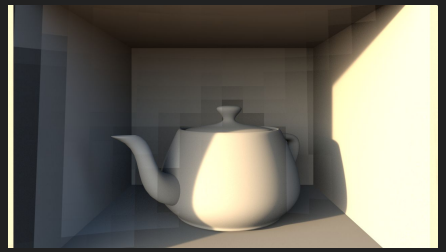
- Veach'95, "Optimally combining sampling techniques for Monte Carlo rendering"
- The illumination from area light sources is computed with two different sampling strategies
  - Based on the light source
  - Based on the BRDF
- When added together, they produce the correct result
  - Like the pieces of a puzzle
- The adaptive dome light changes the balance between two strategies for different regions of the image
  - Eventually leads to less noise and faster renders



Non-adaptive dome

Adaptive dome

Light
sampling



+

BRDF
sampling

=

Final result

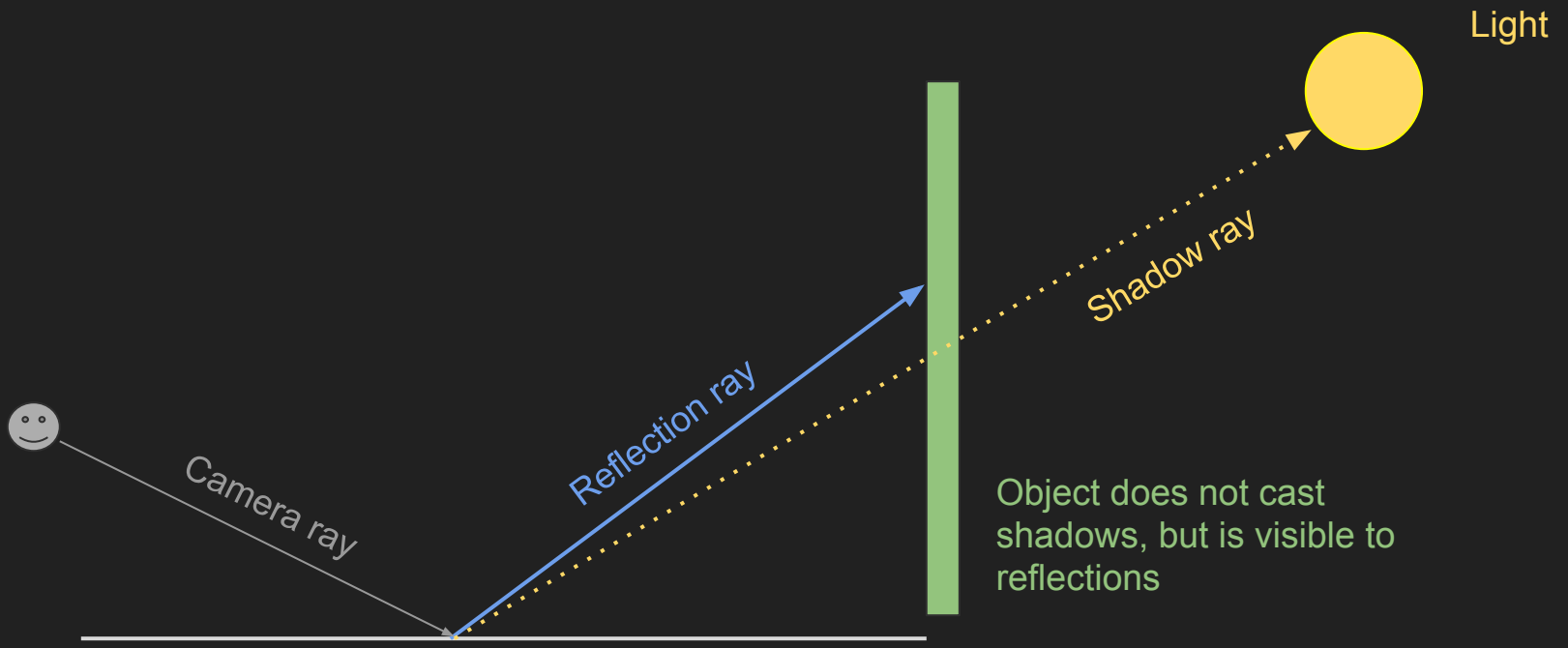
6m 49s

3m 57s

Multiple importance sampling - breaking the balance

- If the balance between light sampling and BRDF sampling is broken we get artifacts
- The balance can be broken for different reasons:
  - Different highlight and reflection glossiness
  - Different GI and shadow visibility
  - Different reflections and shadow visibility
  - Object doesn’t receive shadows
  - Light doesn’t cast shadows
  - Light has include/exclude list (light/shadow linking)
- This wasn’t much of an issue with non-adaptive dome lights
  - Result was still not “accurate”, but was acceptable (no artifacts)

Multiple importance sampling - breaking the balance



Multiple importance sampling - breaking the balance

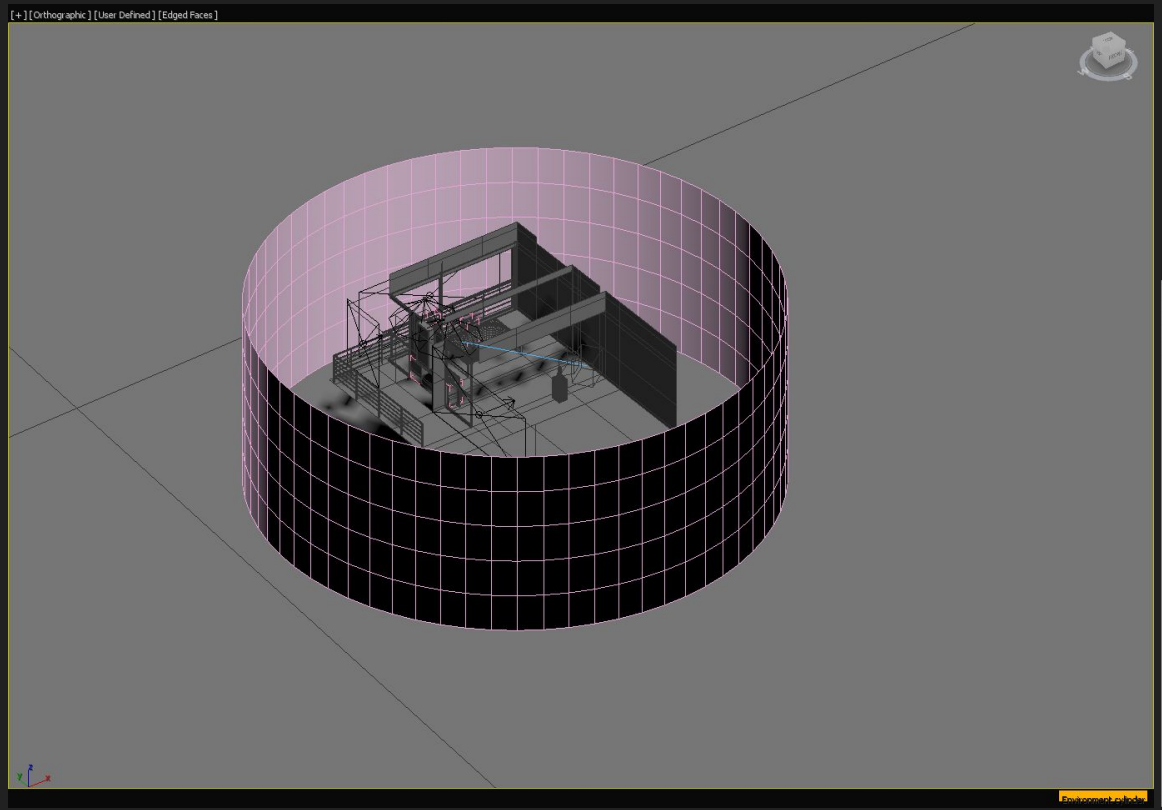
- Breaking the balance is not physically accurate
- But there are still very practical situations where it is extremely useful

Dome light only

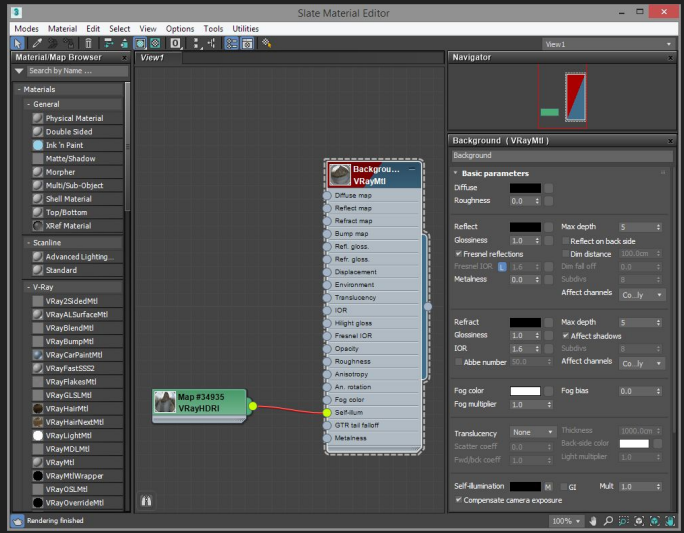
- Lighting is ok
- Background is not what we want though

Environment that we actually want to see

Cylinder (or planes) for the environment



- Usually a VRayMtl with black diffuse and only self-illumination



Regular geometry

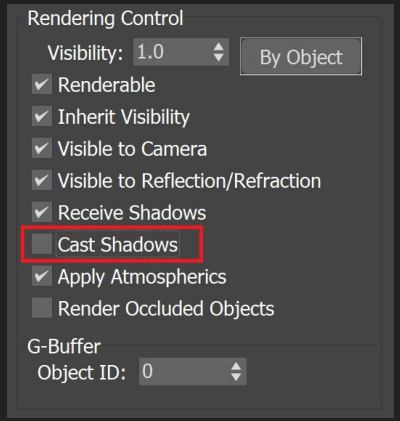
- Lighting is wrong - dome light is blocked
- Background is ok
- Reflections are ok

Visible to camera rays only

- Lighting is ok
- Background is ok
- Reflections are strange

Visible in reflections, cast shadows off

- Lighting is ok
- Background is ok
- Reflections are ok



Dome light only

- Lighting is ok
- Background is not what we want though

Back to the problem

- Objects with inconsistent visibility cause problems
  - Also for Corona’s new light solver
- How to fix this?
  - Warn the user and do nothing?
  - Turn off the adaptive dome light?
  - Abandon the approach altogether?
  - Do something else???
- What do users actually expect to happen?
- The problem with devising new methods to solve problems:
  - Nobody can help you

Solutions

- For beta 3, we implemented a kind of a solution
  - Detect rays going through such inconsistent objects and modify the math to remove the artifacts by using not so optimal MIS weights
  - Worked, but hard to implement on the GPU
  - Caused occasional fireflies
  - Non-optimal samping==slower
- The final solution materialized last Friday when I was looking at the GPU code
  - Basically replace the full specular contribution of the light with that of the environment geometry
  - Requires that the renderer can sample diffuse and specular separately

User reports for blocky artifacts

Fixed artifacts

Just dome light

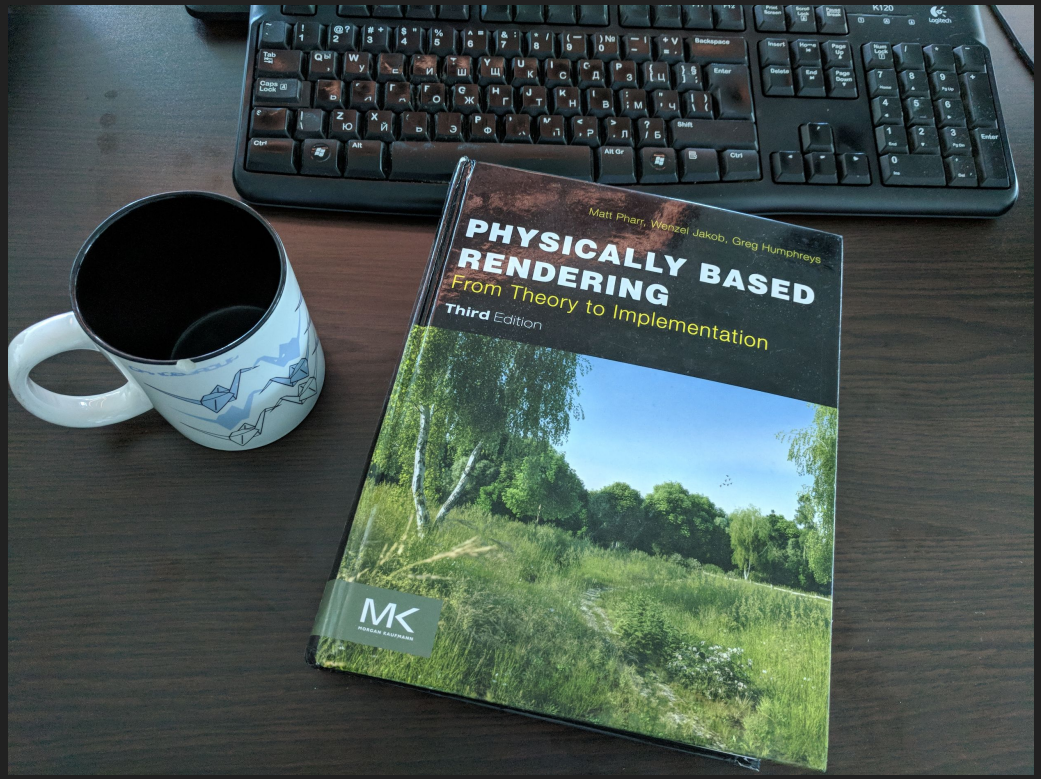
Non-adaptive dome light, 1h 50m 32s

Adaptive dome light, 42m 14s

Mini-research

- When there’s a problem, it’s useful to see how/if others have solved it
  - Not exactly Siggraph paper material
- How do different renderers deal with objects with inconsistent visibility?
- Turns out that there are two ways to handle direct illumination with MIS
  - PBRT-style
  - Mixed style
- This is a fundamental difference between renderers
  - Even if they are all in “brute force” mode and produce otherwise identical results

“Physically Based Rendering” book

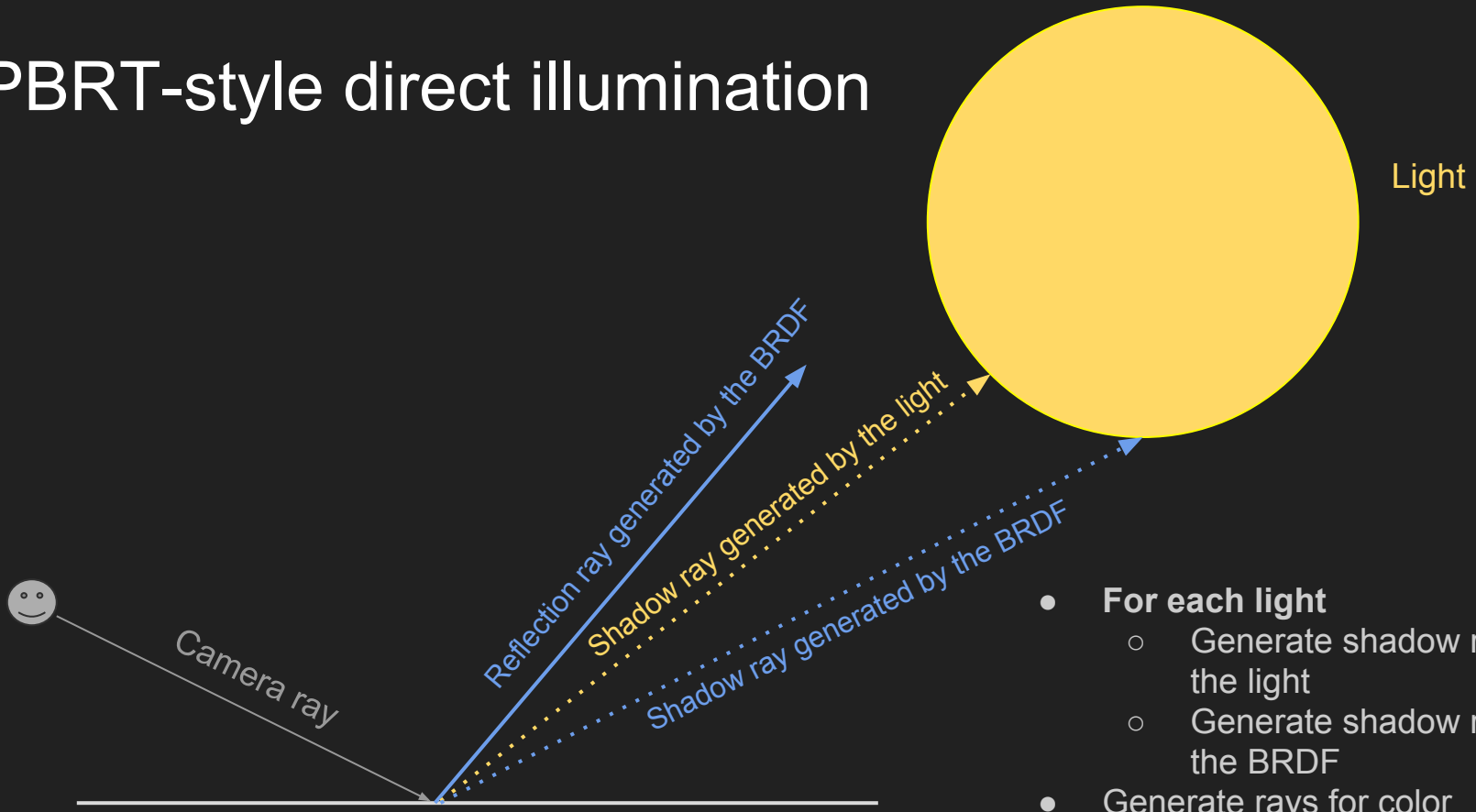


- In photorealistic rendering, when people say “by the book”...
  - ...this is the book.

PBRT-style direct illumination

- PBRT-style
  - When evaluating lights, generate rays both from the lights and the BRDF
    - Trace all of them as shadow rays and then combine the contributions with MIS for the full direct light contribution
  - Then trace additional BRDF rays for diffuse GI (color bleeding) and reflections/refractions
    - Such rays are not affected by light sources
- Light sources may not be a part of the normal scene at all (i.e. always invisible)
- Objects with inconsistent visibility do not cause issues
  - However results might still be different from artists' expectation
- More rays might be traced and the renderer might be slower
  - For each light source, we need to generate and trace rays for the light and the BRDF
  - Then again we need to generate rays for the BRDF to trace GI, reflections and refractions
    - The BRDF is essentially sampled multiple times

PBRT-style direct illumination

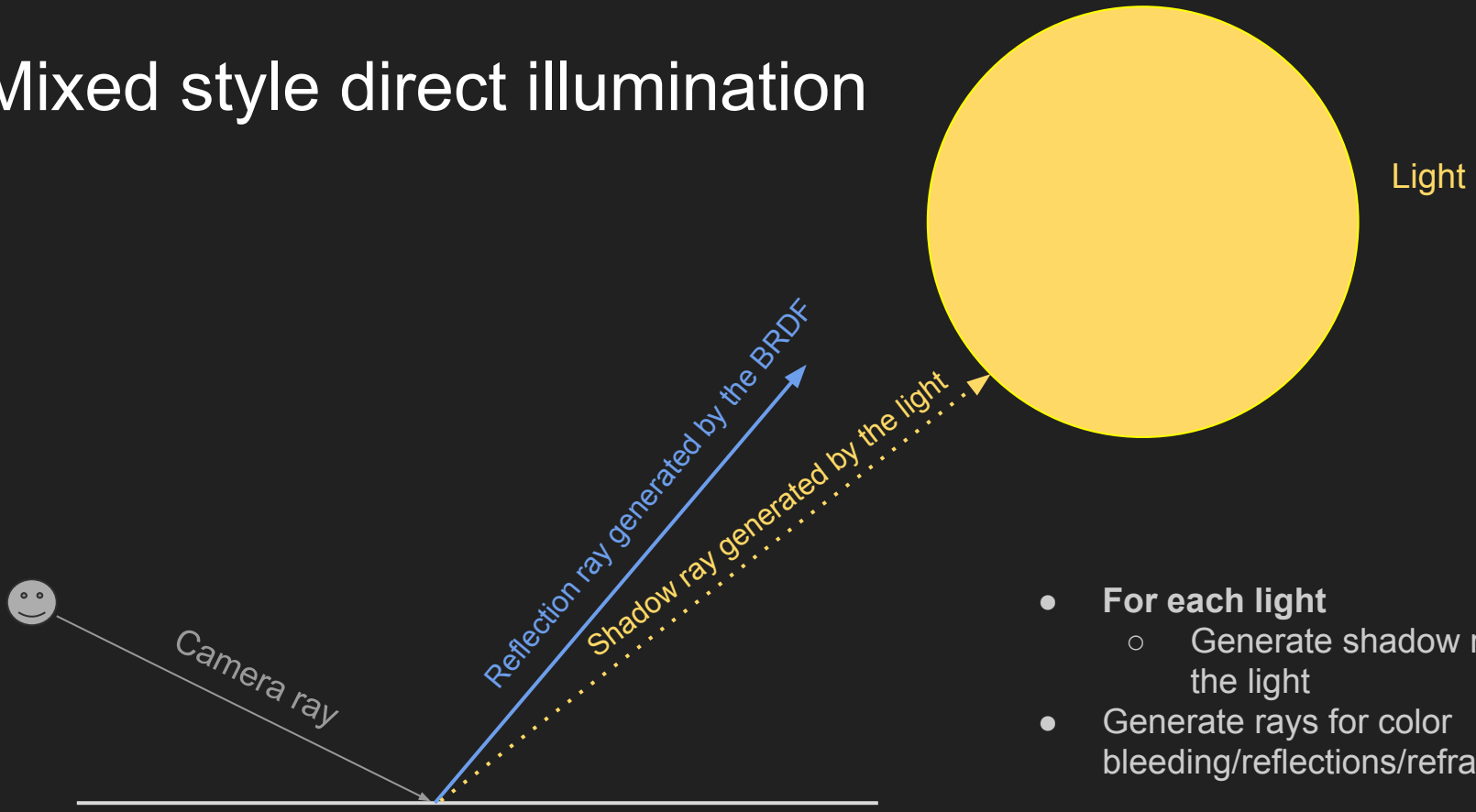


- **For each light**
  - Generate shadow rays from the light
  - Generate shadow rays from the BRDF
- Generate rays for color bleeding/reflections/refractions

Mixed style direct illumination

- Mixed style
  - When evaluating lights, generate rays only for the lights and trace them as shadow rays
    - Only the light part of the MIS contribution of is computed
  - Then trace additional rays for diffuse GI and reflections/refractions
    - If such rays happen to intersect a light source, its BRDF contribution of direct lighting is computed and added to the result
- More care must be taken to ensure objects with inconsistent visibility do not cause issues
- Typically less rays need to be traced to produce the same result

Mixed style direct illumination

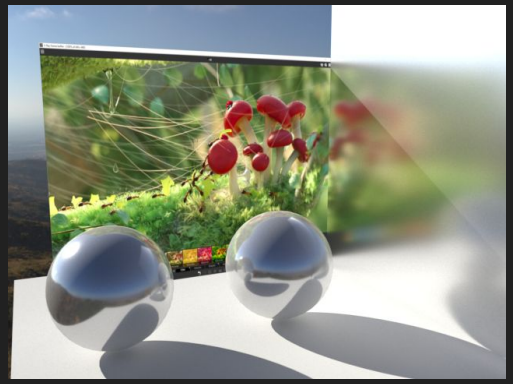
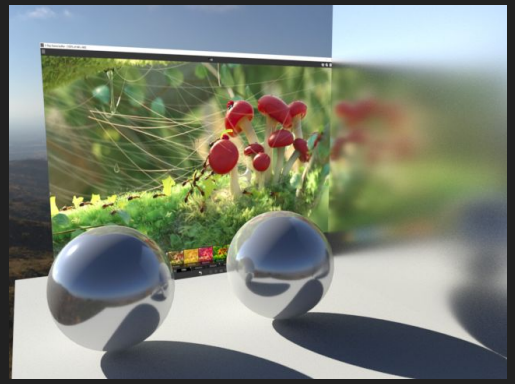
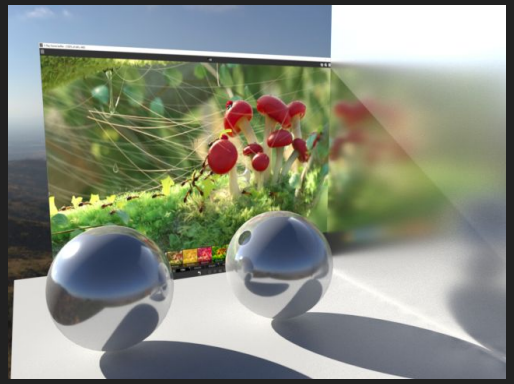
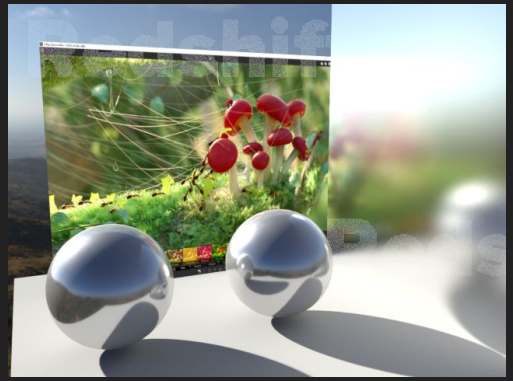


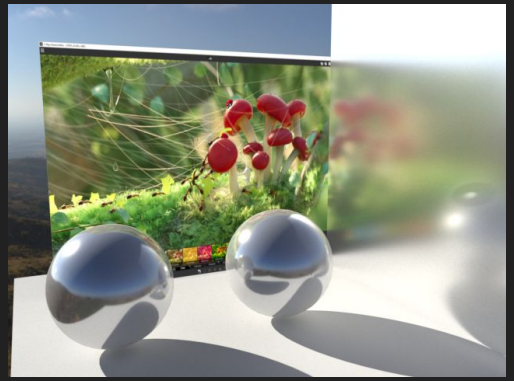
- **For each light**
  - Generate shadow rays from the light
- Generate rays for color bleeding/reflections/refractions

Examples

- PBRT-style renderers
  - Arnold
  - RenderMan
  - PBRT
- Mixed style
  - V-Ray
  - Corona

Comparing renderers



Questions?