Hardware Implementation

SIGGRAPH2007

Pascal Gautron

Post-Doctoral Researcher

France Telecom R&D Rennes

France

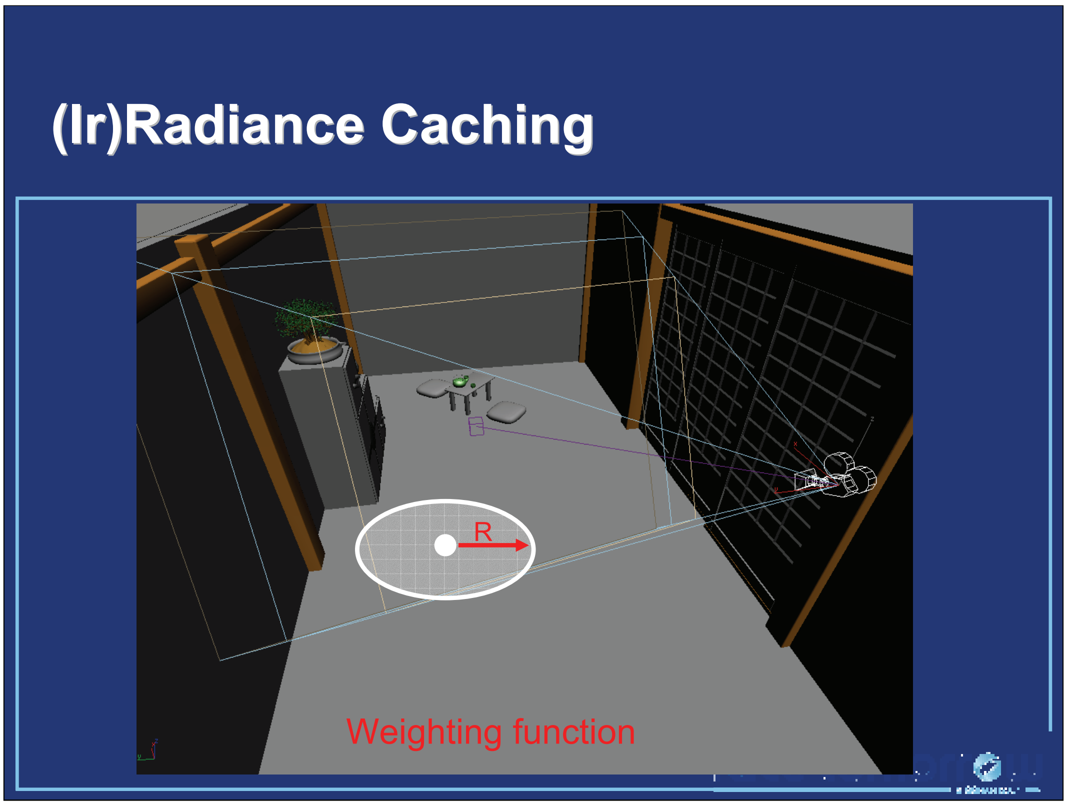
Radiance Cache Splatting

Pascal Gautron, Jaroslav Krivanek, Kadi Bouatouch, Sumanta Pattanaik

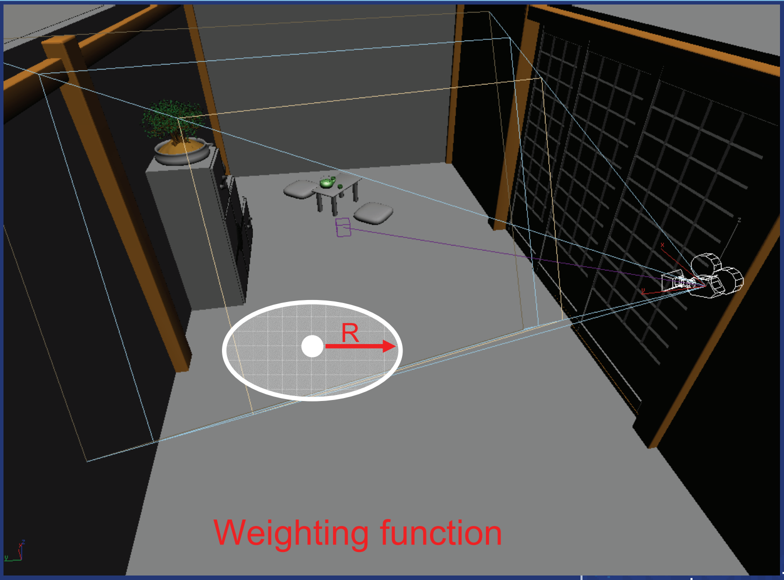
In Proceeding of Eurographics Symposium on Rendering 2005

Course Outline

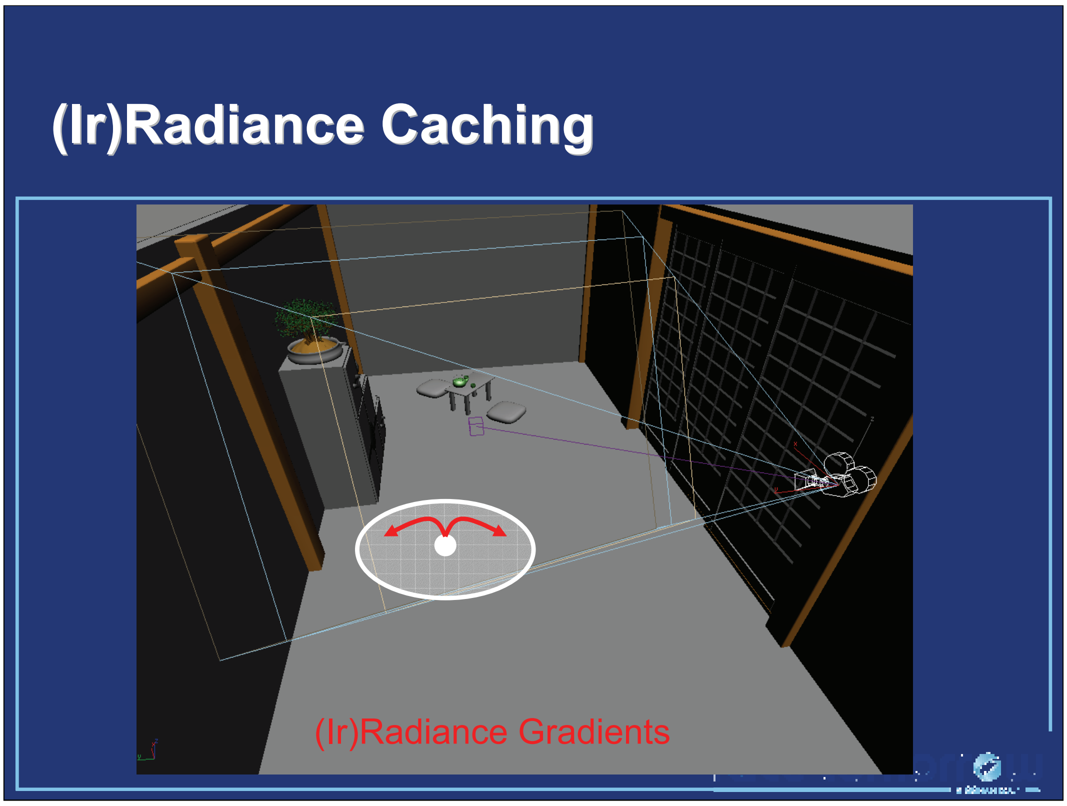
- Irradiance Caching: ray tracing and octrees
- A reformulation for hardware implementation
- Irradiance Cache Splatting
- GPU-Based hemisphere sampling



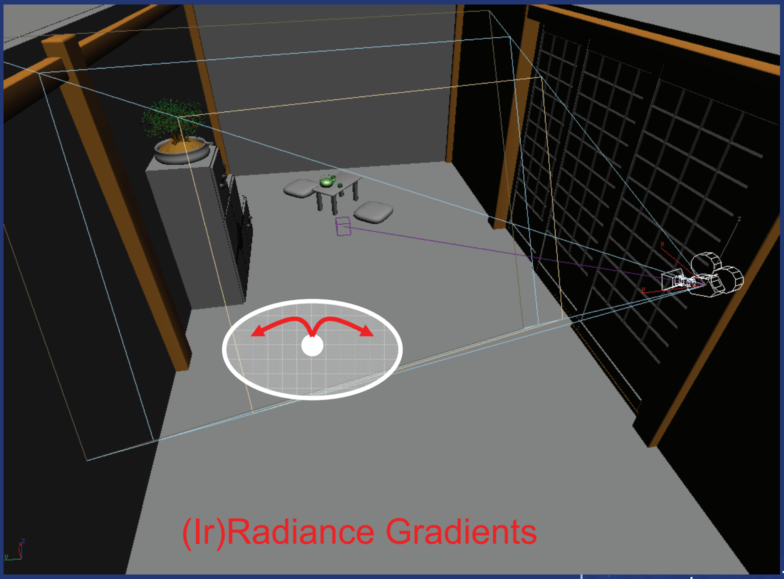
(Ir)Radiance Caching



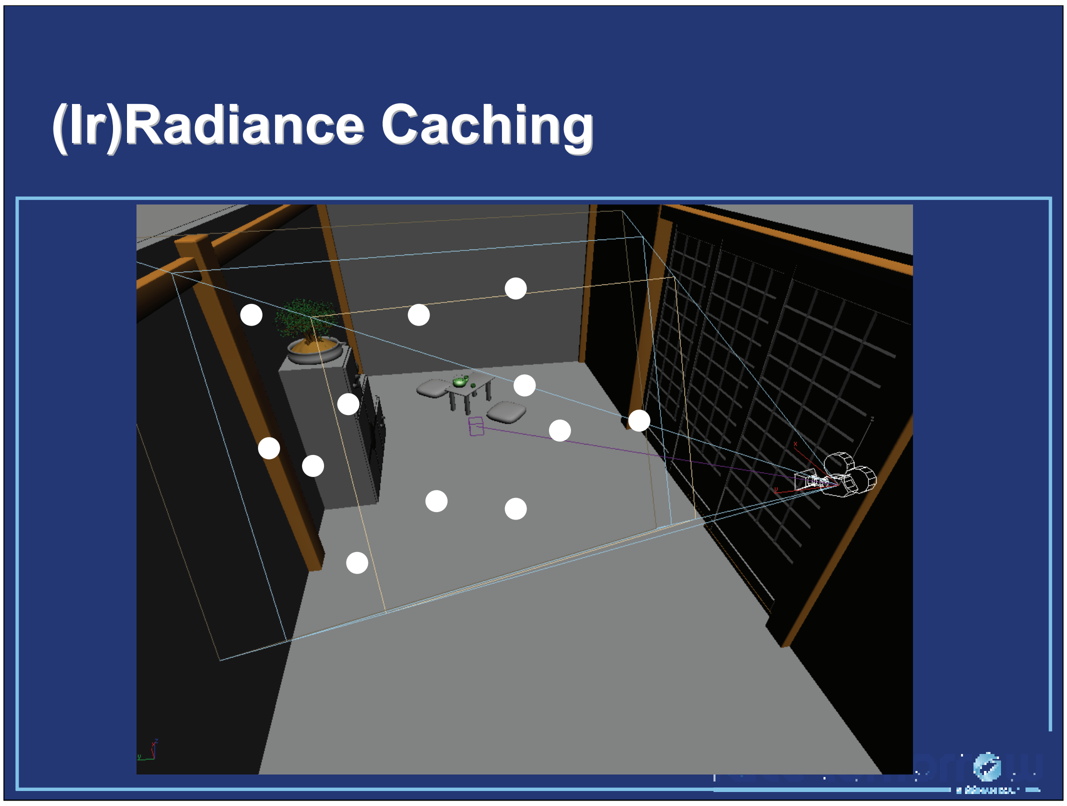
The irradiance caching algorithm is based on sparse sampling and interpolation of indirect diffuse lighting at visible points. Each irradiance record contributes to the indirect lighting of points within its zone of influence. The size of this zone is adapted according to the mean distance R to the surrounding objects using the irradiance weighting function [Ward88].



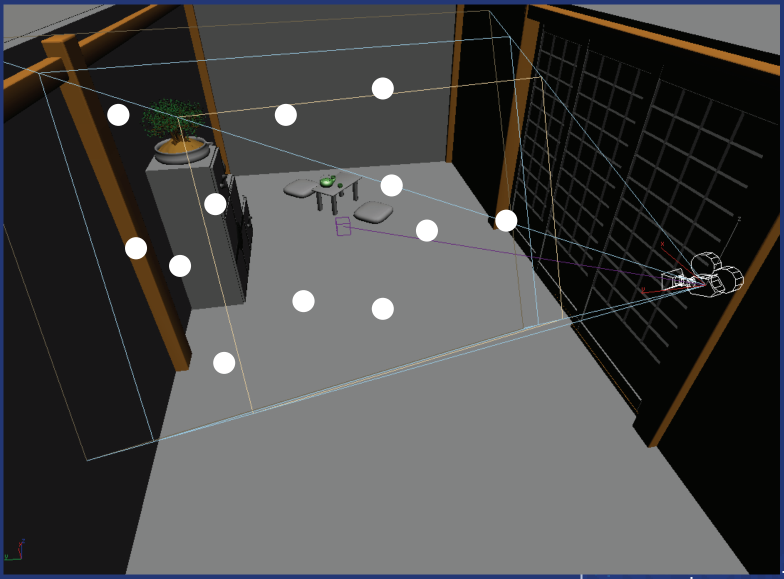
(Ir)Radiance Caching



The irradiance value at points within the zone of influence of a record can be extrapolated using irradiance gradients [Ward92, Krivanek05a, Krivanek05b].



(Ir)Radiance Caching



When the zones of influence of the records cover the entire zone visible from the viewpoint, the image representing the indirect lighting can be rendered. The irradiance caching algorithm is then divided into three steps:

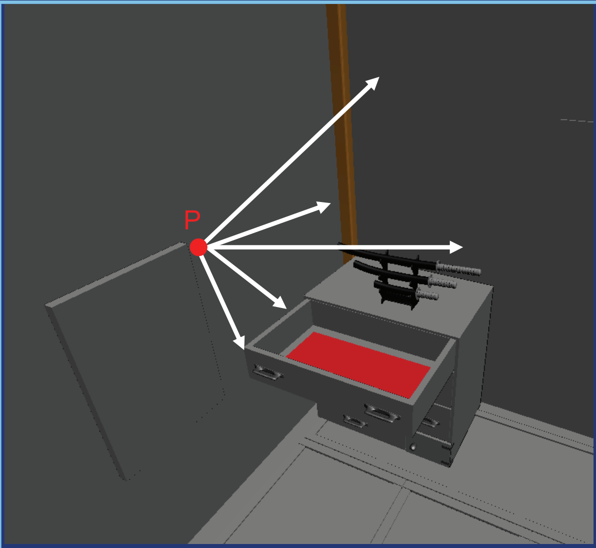
-The computation of the records

-The records storage

-The estimation of the indirect lighting using nearby records

Record Computation

Ray Tracing



$$E(P) = \int L_i(P, \omega_i) * \cos(\theta) d\omega_i$$

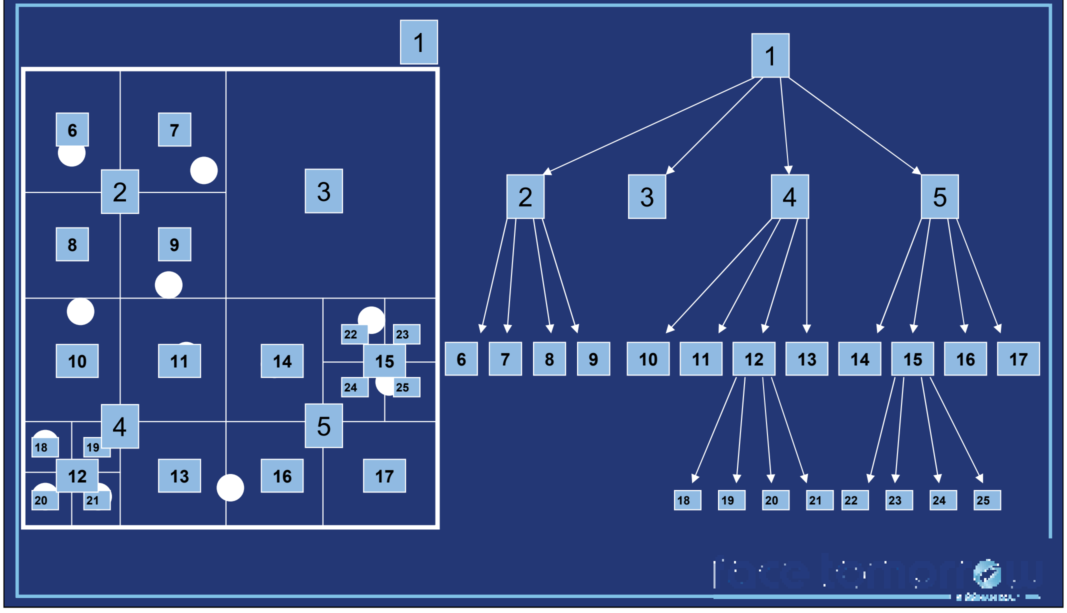
Monte Carlo Integration

The computation of the irradiance value at a given point P requires the evaluation of the integral of the lighting over the surrounding hemisphere. This integral is typically estimated by Monte Carlo integration. Since the irradiance caching algorithm reuses the value of irradiance records over many pixels, the irradiance value of the record is computed with high precision, typically by tracing several hundreds to thousands rays.

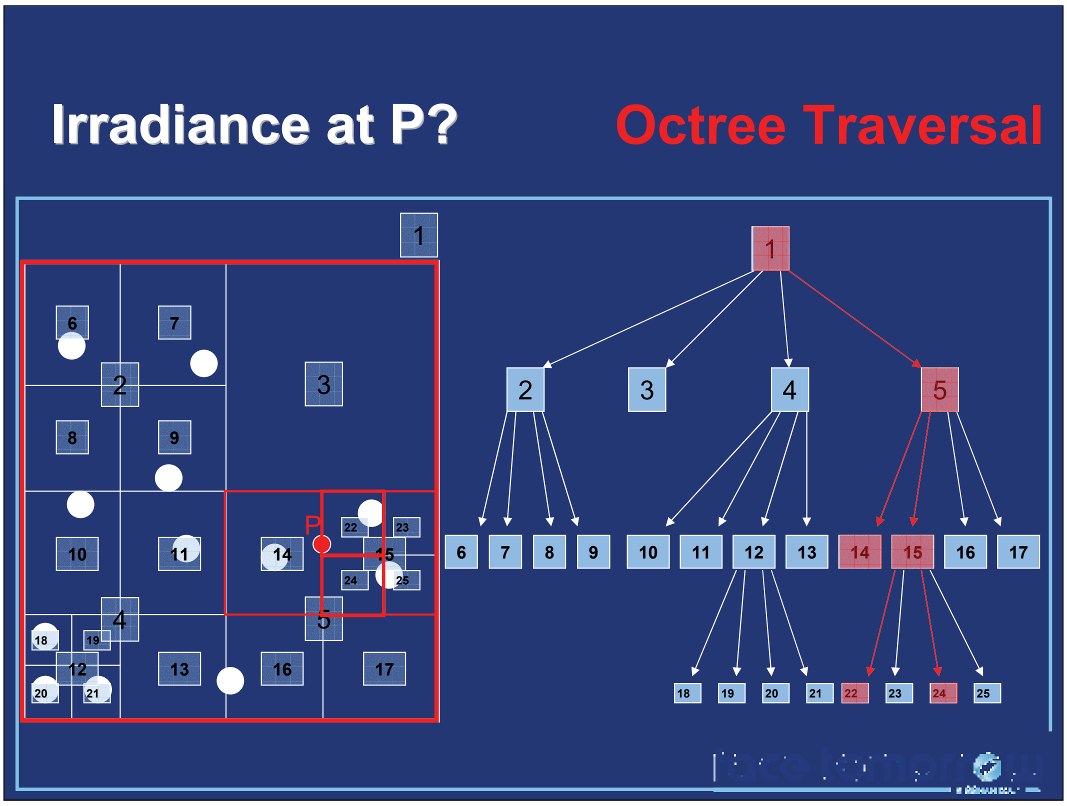
Once the irradiance value is computed, a record is created. This record contains the corresponding position, normal, irradiance, gradients, and mean distance to the surrouding objects.

Cache Storage

Octree

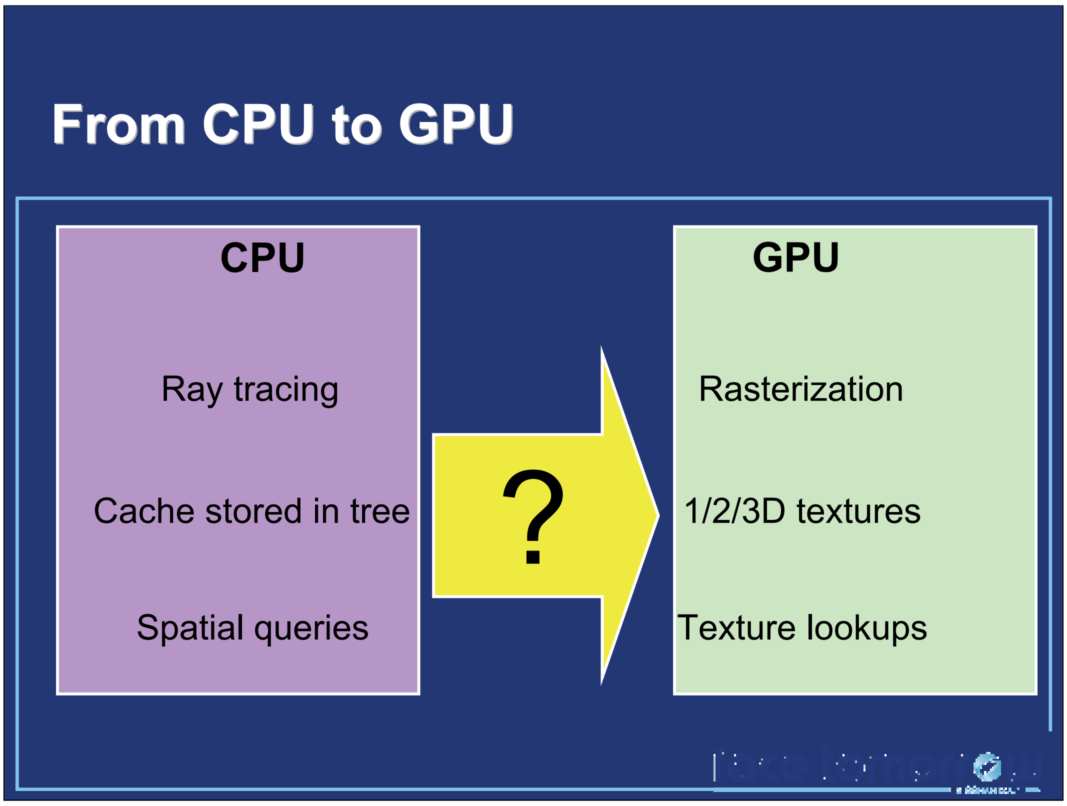


The records are then stored in an octree, which allows for fast spatial queries. Note that the octree is a recursive data structure, which construction involves many conditional statements.



Irradiance at P? Octree Traversal

The irradiance at a point P can then be estimated efficiently by querying the octree for nearby records. However, this involves a traversal of the structure, which also involves many conditional statements.



From CPU to GPU

CPU	GPU
Ray tracing	Rasterization
Cache stored in tree	1/2/3D textures
Spatial queries	Texture lookups

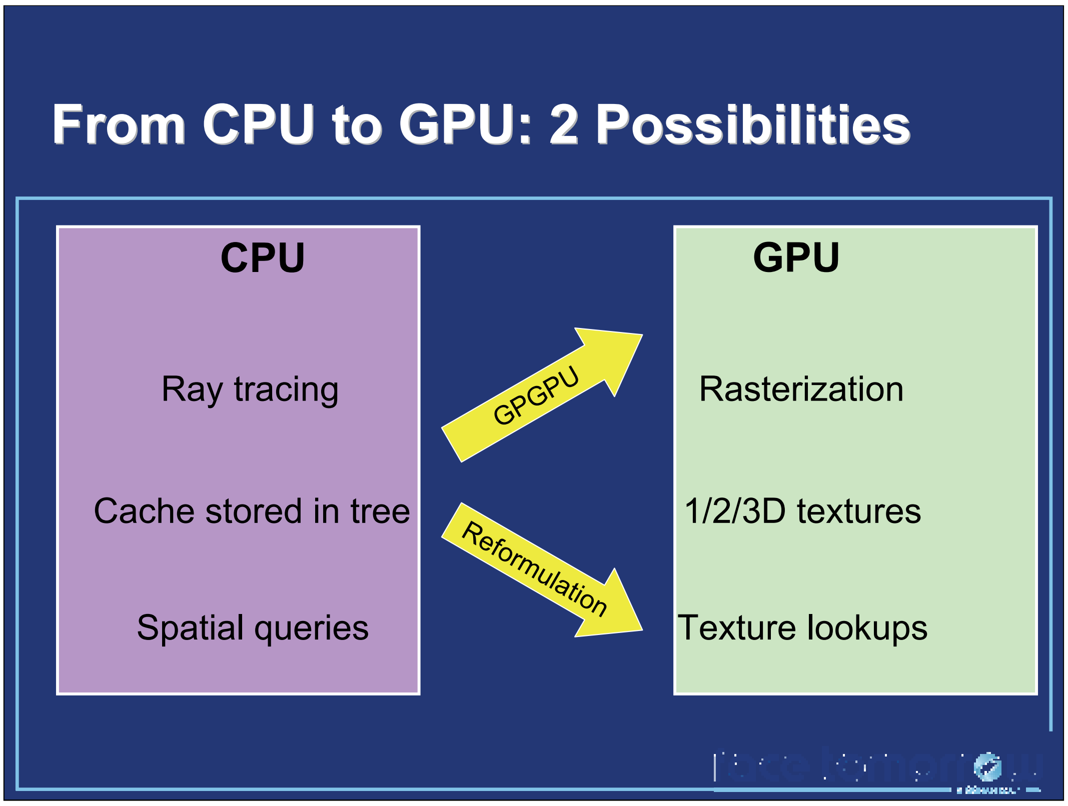
To summarize, the classical irradiance caching algorithm is implemented on the CPU, and is based on:

-Ray tracing

-Octrees

-Spatial queries in the octree

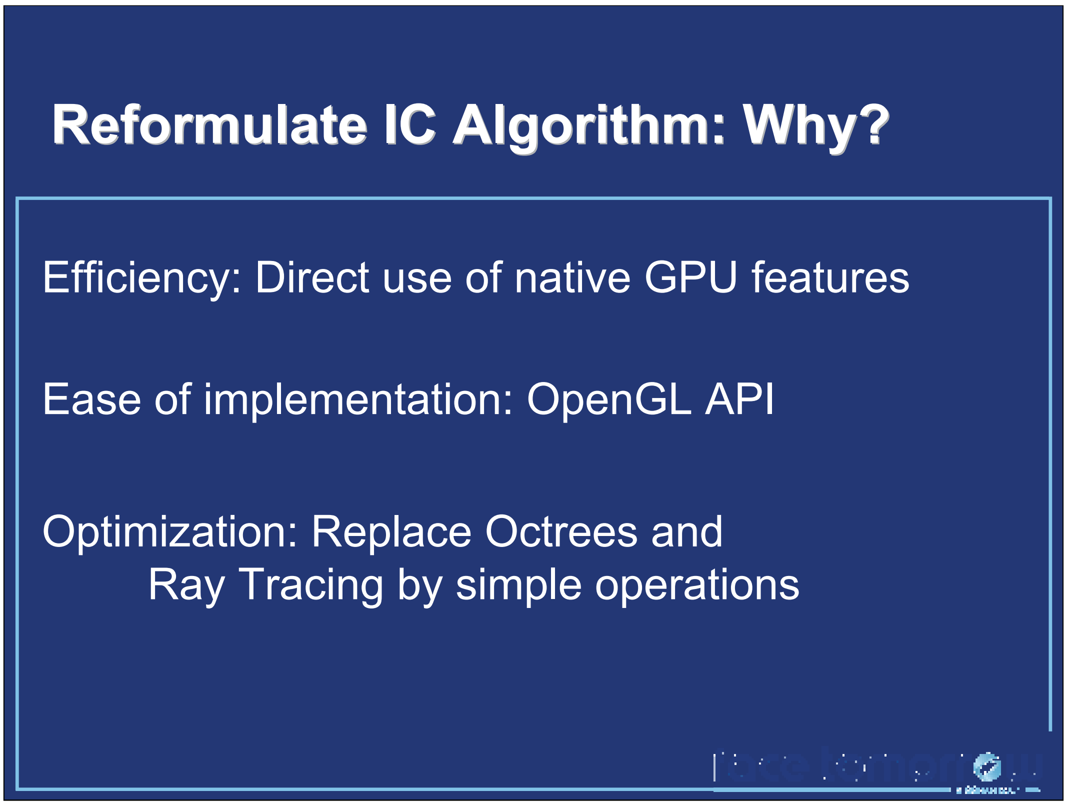
However, the GPU does not natively implement those operations: the visibility tests are performed using rasterization and Z-Buffering, and the only data structures available are 1, 2, and 3D textures. Particularly, the GPUs do not support pointers, which are the most common way of implementing recursive data structures.



From CPU to GPU: 2 Possibilities

The irradiance caching algorithm cannot be directly mapped to the GPU architecture. Two methods can be considered: first, the use of libraries for general purpose computations on the GPU (such as Brook for GPU, http://graphics.stanford.edu/projects/brookgpu/). An important amount of research work has been performed to achieve interactive ray tracing on GPUs (such as [Purcell02, Purcell03]). A kD-Tree implementation for GPUs has also been proposed [Foley05].

An other way of performing irradiance caching on graphics hardware is to reformulate the algorithm so that only the native features of the GPU are used. This would allow us to get the best performance out of the graphics processors.



Reformulate IC Algorithm: Why?

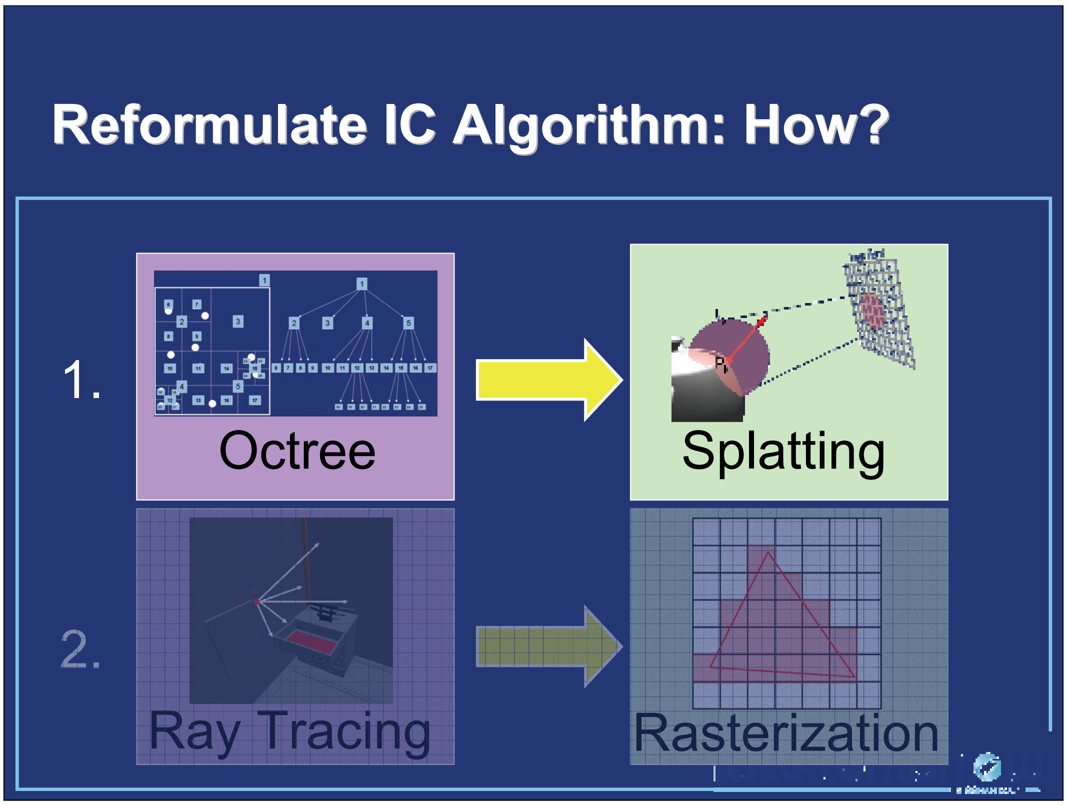
Efficiency: Direct use of native GPU features

Ease of implementation: OpenGL API

Optimization: Replace Octrees and
Ray Tracing by simple operations

More precisely, we chose to reformulate the irradiance caching algorithm for three reasons. First, the direct use of native GPU features allow us to use each part of the GPU at its best, improving the performance. Second, the reformulated algorithm can be implemented directly using at 3D graphics API such as OpenGL or DirectX.

Third, this reformulation gives us the occasion of attacking two costly aspects of the irradiance caching algorithm: the hierarchical data structure, and the irradiance computation using ray tracing. Replacing those by more simple operations on the GPU increases the performance, yielding interactive frame rates in simple scenes.

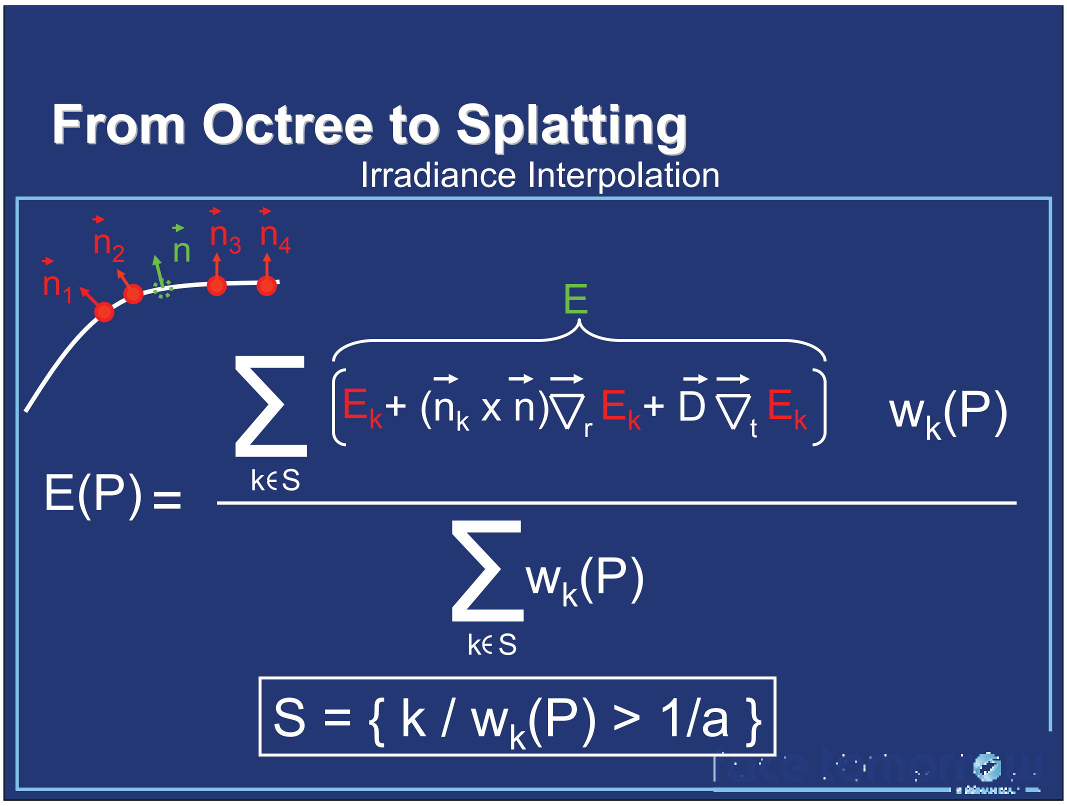


Reformulate IC Algorithm: How?

1. Octree → Splatting

2. Ray Tracing → Rasterization

The reformulation is divided into two tasks: the replacement of the octree by a splatting operation, and the use of rasterization in place of ray tracing.


$\vec n_1$
$\vec n_2$
$\vec n$
$\vec n_3$
$\vec n_4$

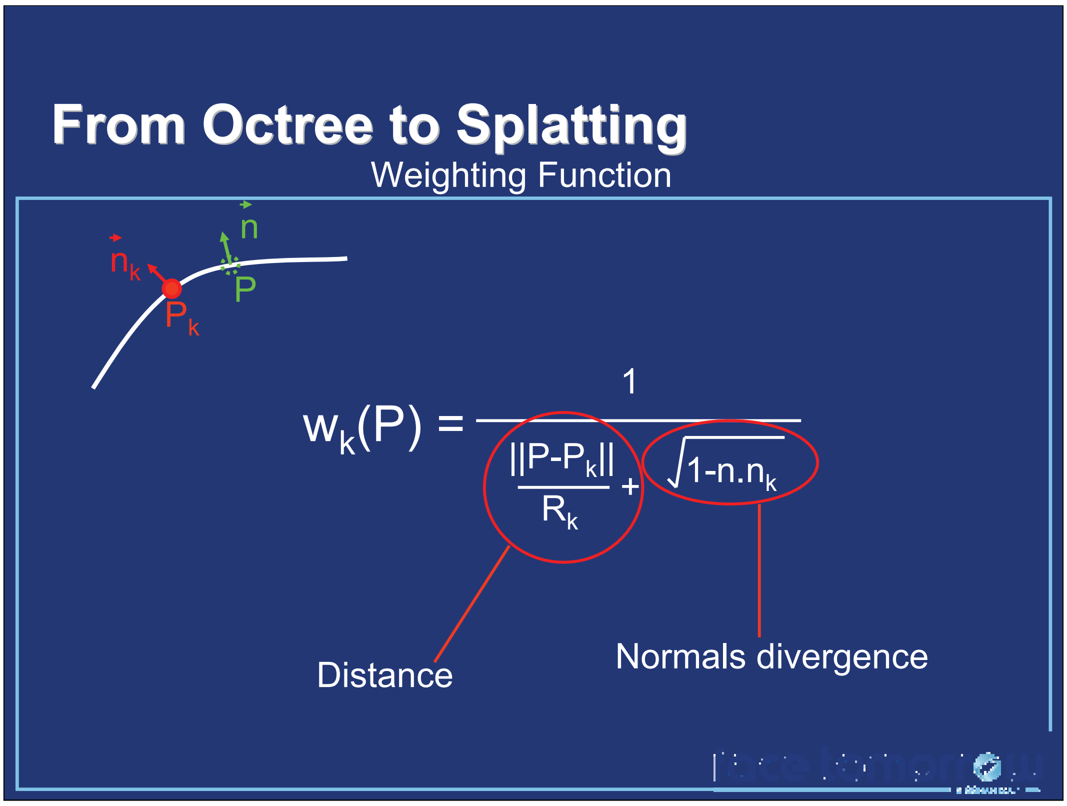
From Octree to Splatting

Irradiance Interpolation

$$E(P) = \frac{\sum_{k \in S} \overbrace{\left[E_k + (\vec{n}_k \times \vec{n})\vec{\nabla}_r E_k + \vec{D}\,\vec{\nabla}_t E_k\right]}^{E} \; w_k(P)}{\sum_{k \in S} w_k(P)}$$

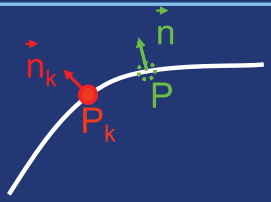
$$\boxed{S = \{\, k \,/\, w_k(P) > 1/a \,\}}$$

The reformulation is based on the irradiance interpolation equation [Ward88, Ward92] presented before: the irradiance estimate at a point P is the weighted average of the contributions of the surrounding records. The contribution of each record is computed using irradiance gradients for translation and rotation. The set S of contributing records is defined as the set of records for which the weigthing function evaluated at P is above a user-defined threshold a.



From Octree to Splatting

Weighting Function



$$w_k(P) = \frac{1}{\frac{\|P-P_k\|}{R_k} + \sqrt{1-n.n_k}}$$

Distance

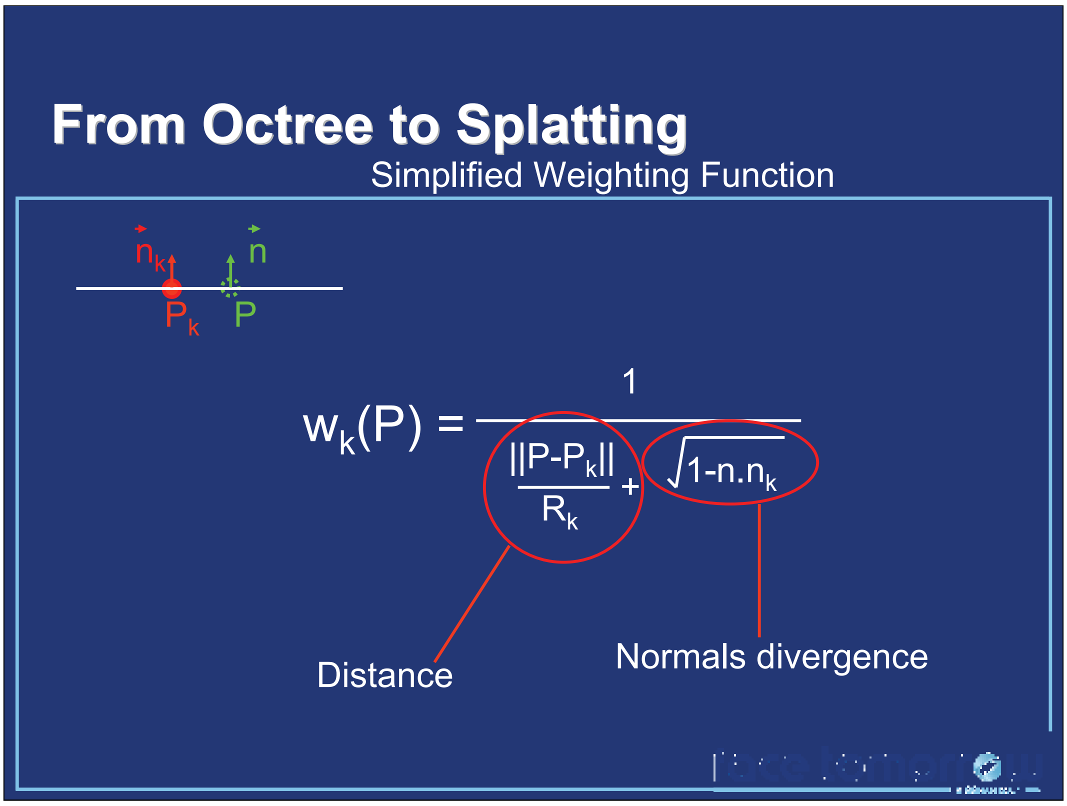
Normals divergence

The weighting function is very simple, and depends on:

-The distance between the record location and the point P

-The mean distance R to the surrounding objects

-The divergence of the surface normals between the record location and the point P



From Octree to Splatting

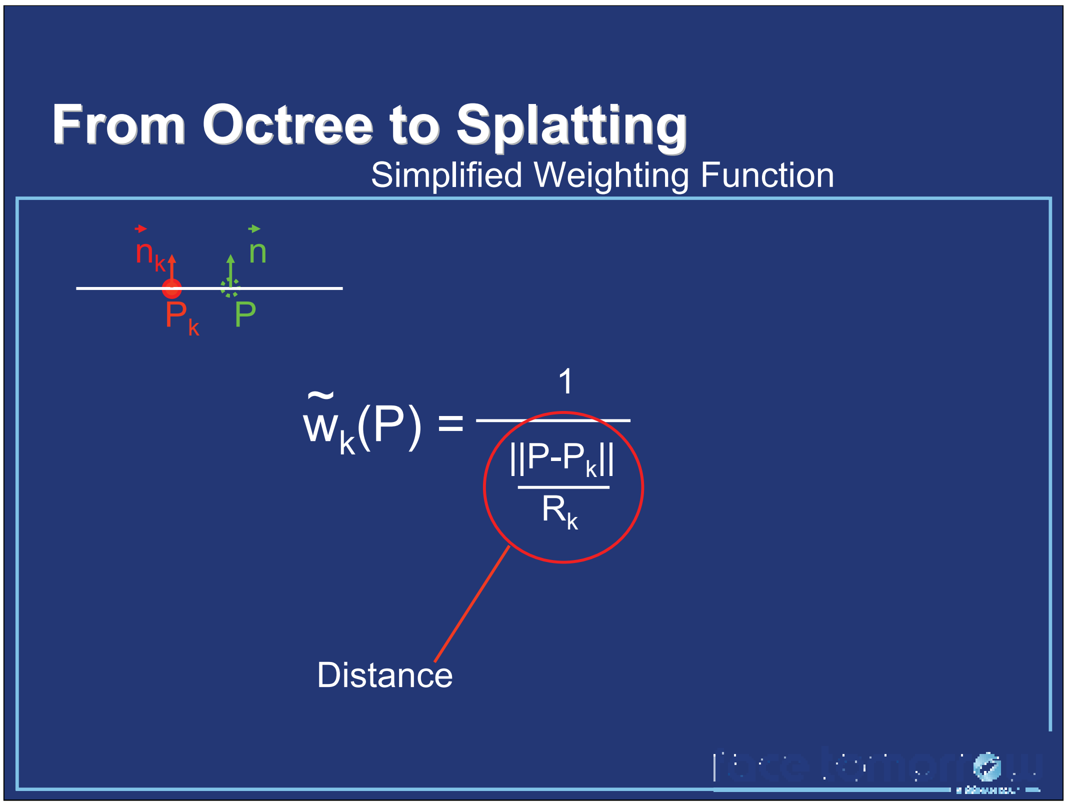
Simplified Weighting Function

$\vec{n}_k$
$\vec{n}$
P_k
P

$$w_k(P) = \frac{1}{\frac{\|P-P_k\|}{R_k} + \sqrt{1-\vec{n}.\vec{n}_k}}$$

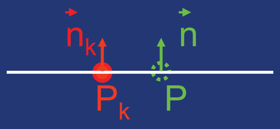
Distance
Normals divergence

If we assume that the record location and the point P **always have the same normal**, the weighting function can be simplified by removing the dependence to the surface normals.



From Octree to Splatting

Simplified Weighting Function



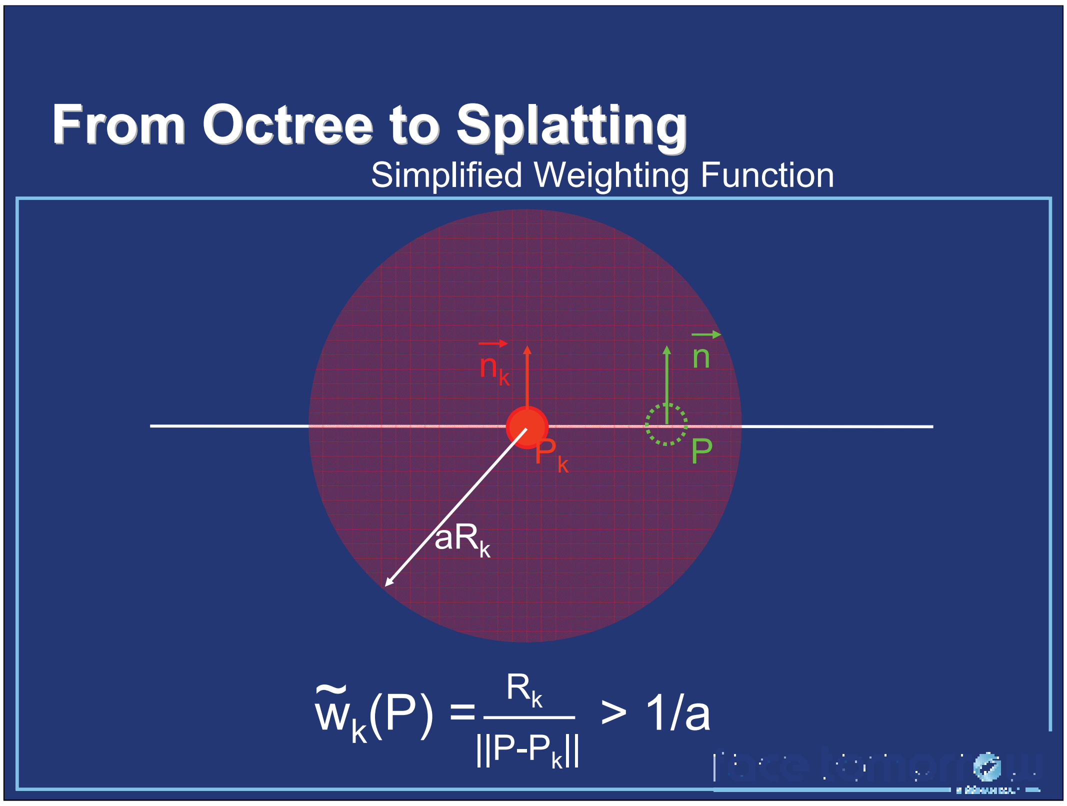
$$\tilde{w}_k(P) = \frac{1}{\frac{\|P-P_k\|}{R_k}}$$

Distance

This yields a simplified weighting function, which depends only on two factors:

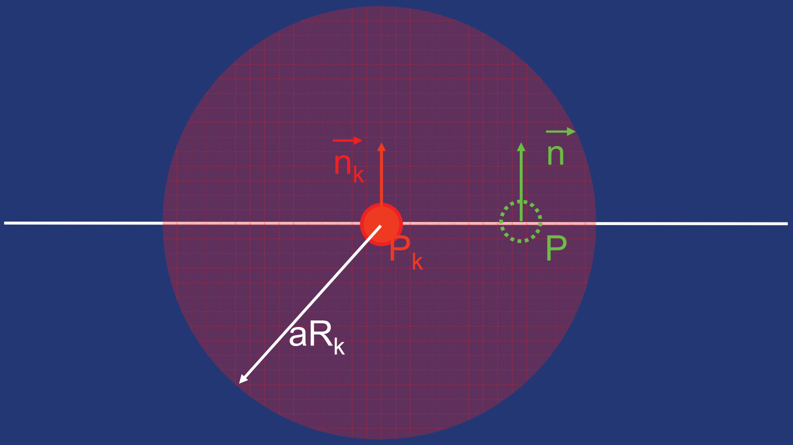
-The distance between the record location and the point P

-The mean distance to the surrounding objects



From Octree to Splatting

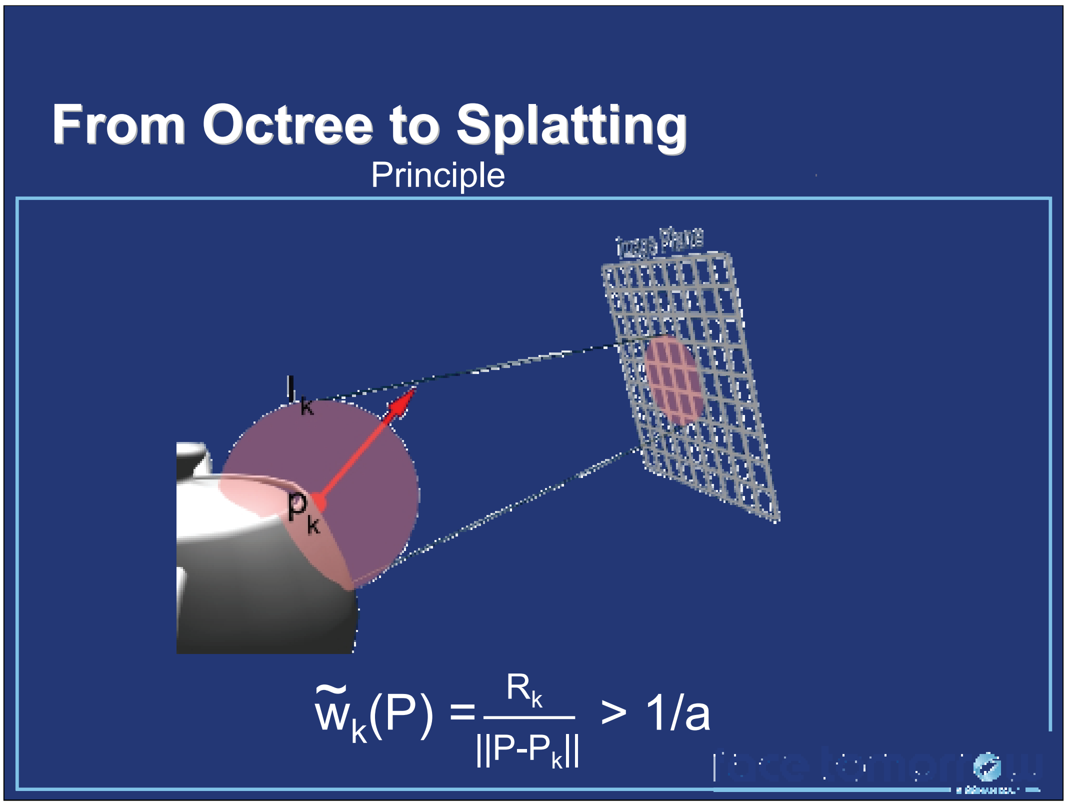
Simplified Weighting Function



$$\tilde{w}_k(P) = \frac{R_k}{||P-P_k||} > 1/a$$

Using this simplified weighting function, a record k contributes to **all** points located within a sphere centered at the record location, with radius aRk. Hence this radius depends not only on the user-defined parameter a, but also on the mean distance to the surrounding objects.

Note that the simplified weighting function removes the constraint on the surface normals. Therefore, the set of points at which the record actually contributes (with respect to the full weighting function) is a subset of the points located within the sphere.

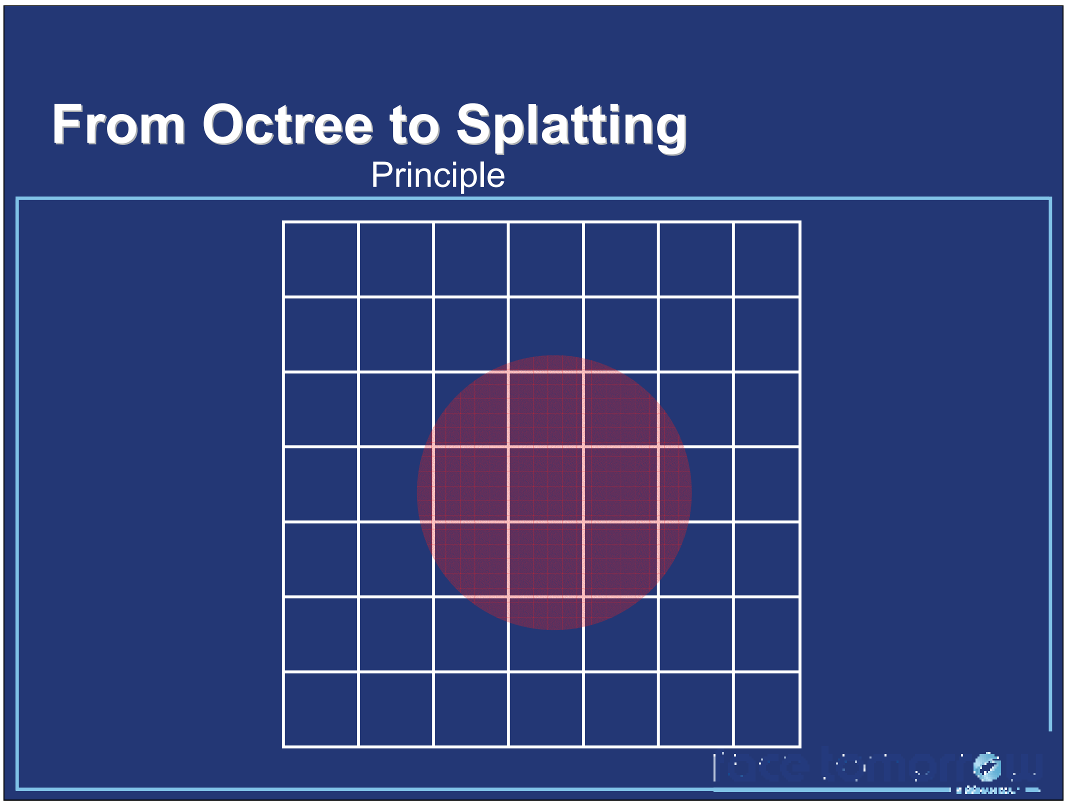


From Octree to Splatting

Principle

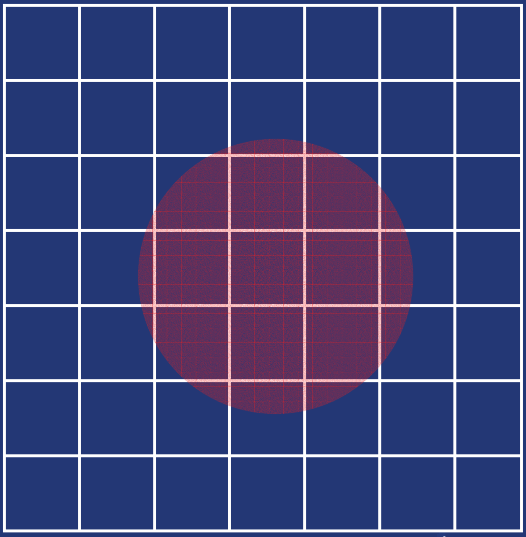
$$\tilde{w}_k(P) = \frac{R_k}{\|P-P_k\|} > 1/a$$

The sphere can be splatted onto the image plane. Hence the covered pixels correspond to the visible points at which the record may contribute.



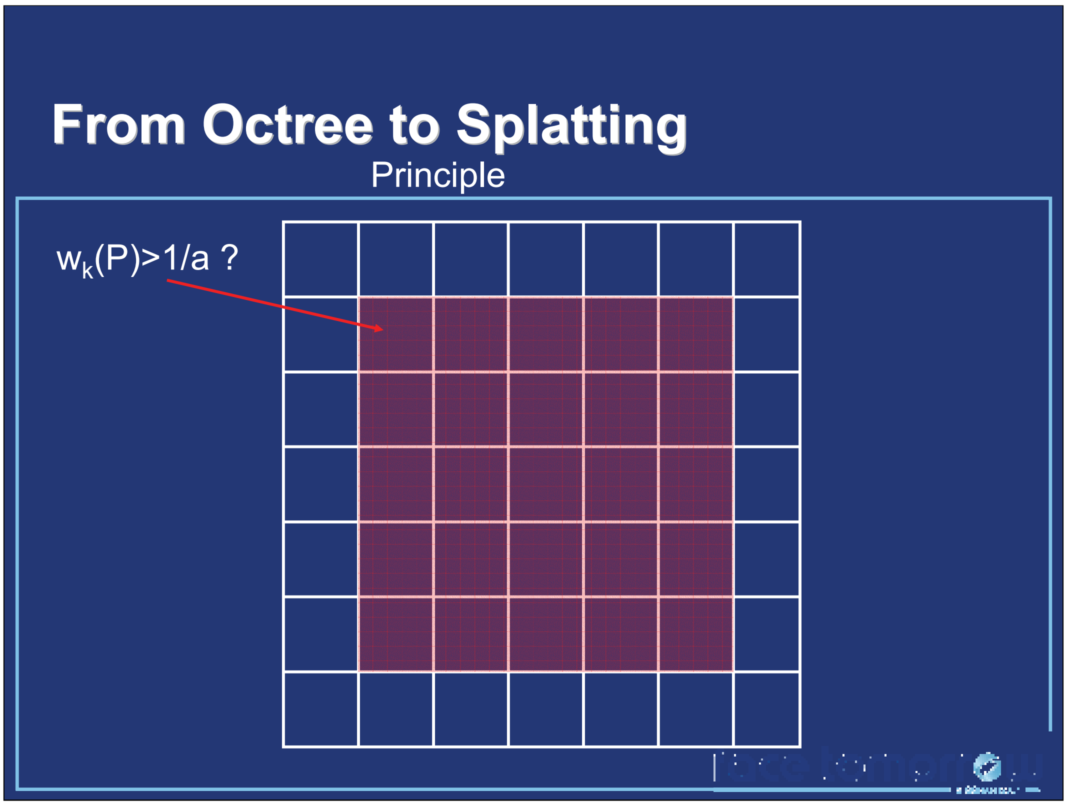
From Octree to Splatting

Principle



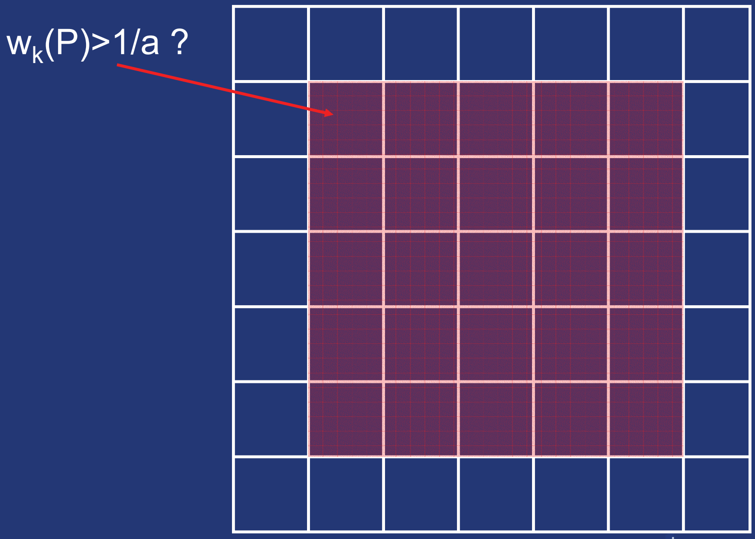
Let us consider the image plane, on which the sphere has been splatted. The splatted sphere encloses **all** the visible points at which the considered record may contribute (with respect to the full weighting function). Our goal is now to select the points for which the condition on the **full** weighting function is satisfied, that is

wk(P) > 1/a .

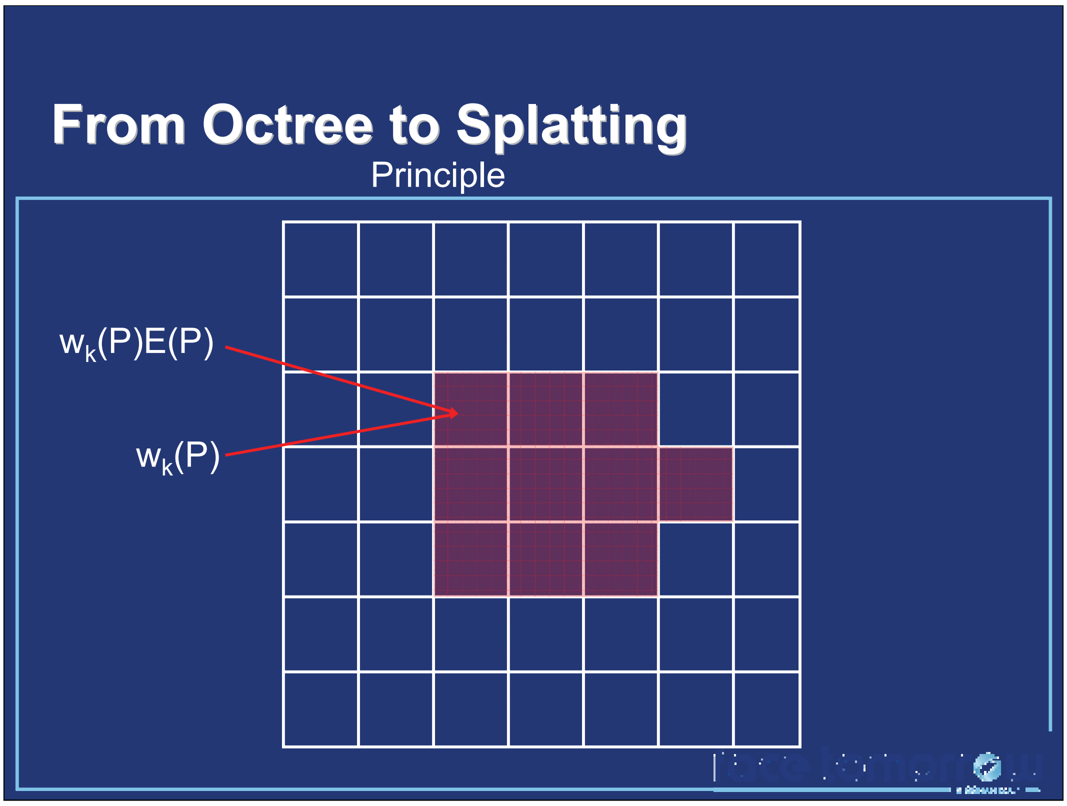


From Octree to Splatting

Principle



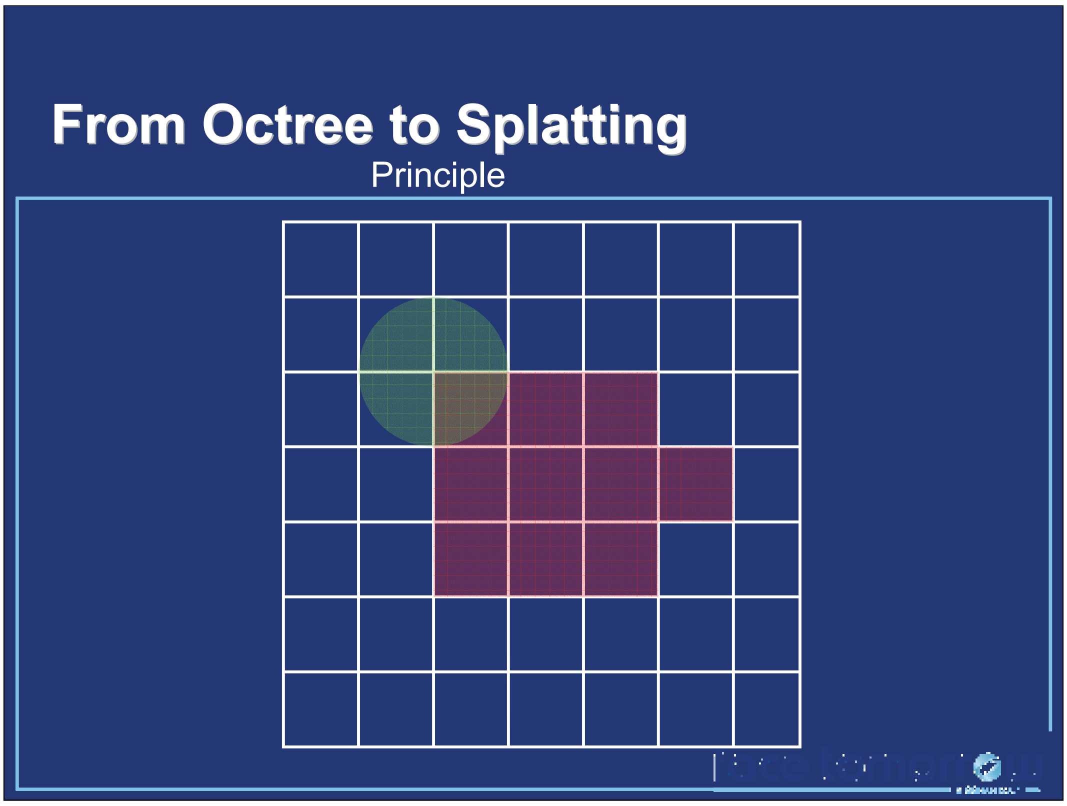
For the convenience of implementation, we use a quadrilateral tightly enclosing the splatted sphere. For each point visible through the pixels of the quadrilateral, the full weighting function is evaluated, and tested against the user-defined threshold. If the condition is not satisfied, the pixel is discarded.



From Octree to Splatting

Principle

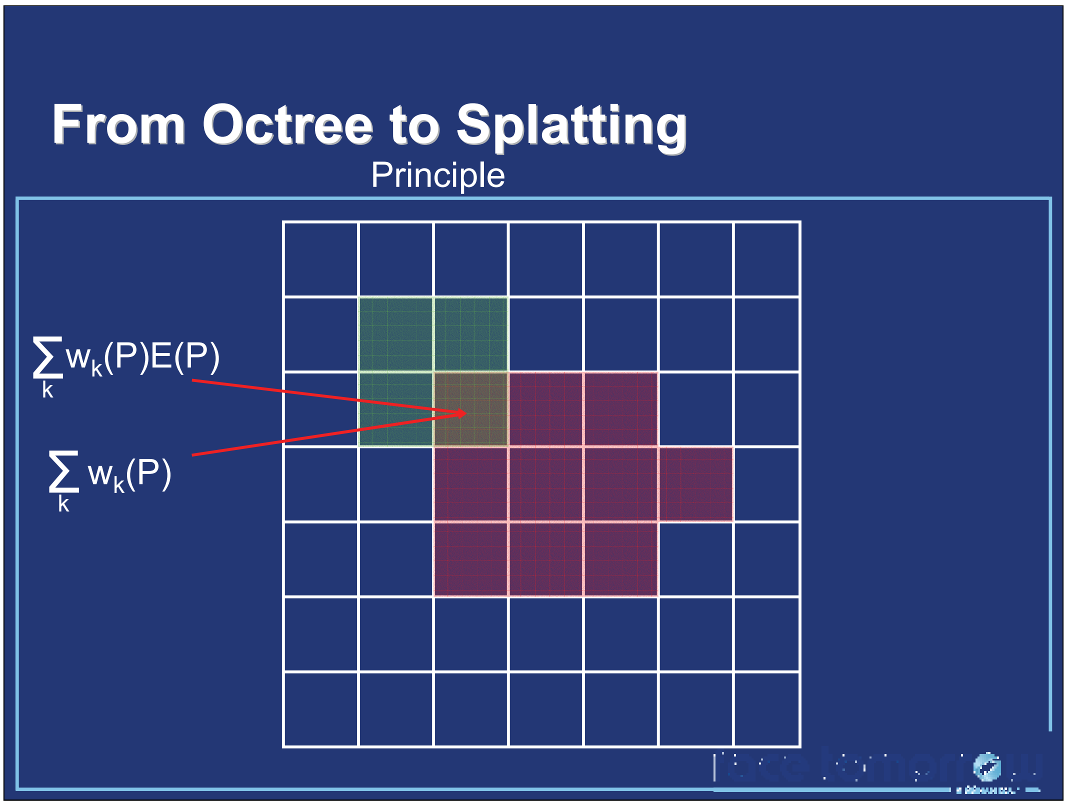
This yields a set of pixels, corresponding to the set of visible points at which the record actually contributes. At those points, we compute separately the weighted contribution of the record (with respect to the full weighting function and to the irradiance gradients) and the weight of the contribution. This information can be easily stored within floating point RGBA pixels, using RGB for the weighted contribution, and the alpha channel for the weight value.



From Octree to Splatting

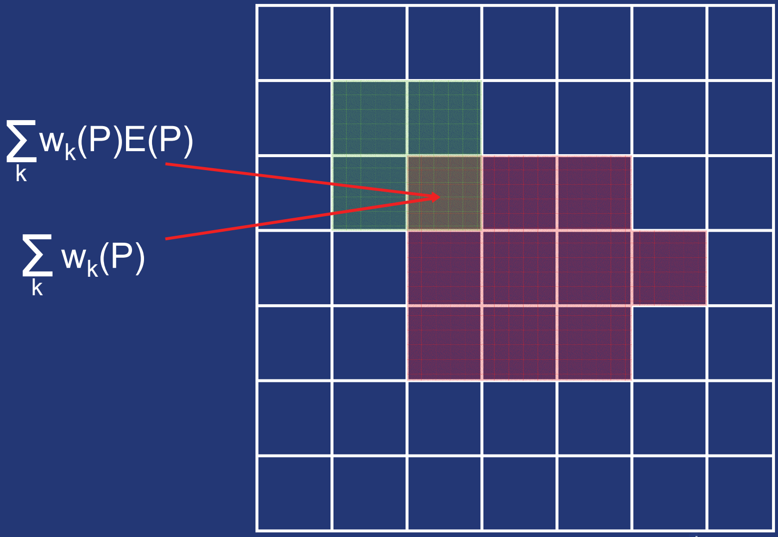
Principle

When splatting an other record, the same operations are to be done. However, the zones of influence of the two records overlap.



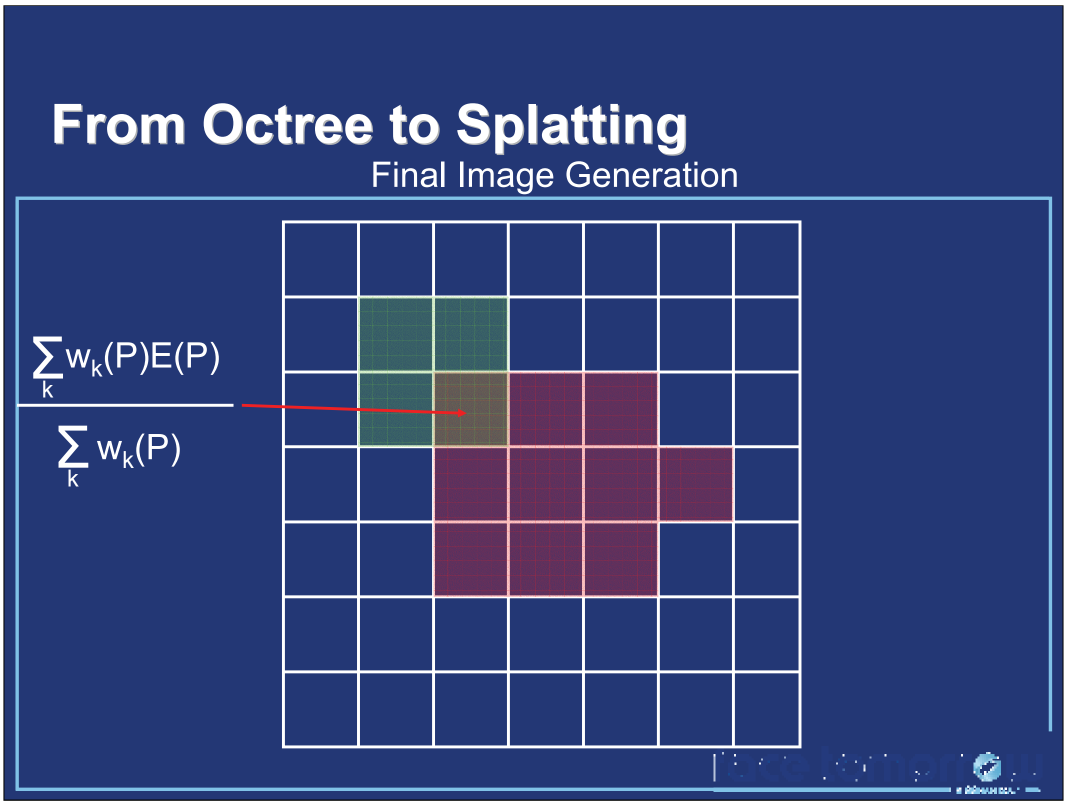
From Octree to Splatting

Principle



In this case, we first compute the weighted contribution and weight of the second record as described before. Then, the built-in alpha blending of graphics processors adds the contributions and weights together in the overlapping area.

Then, each pixel contains both the weighted sum of the contributions, and the sum of the contribution weights.

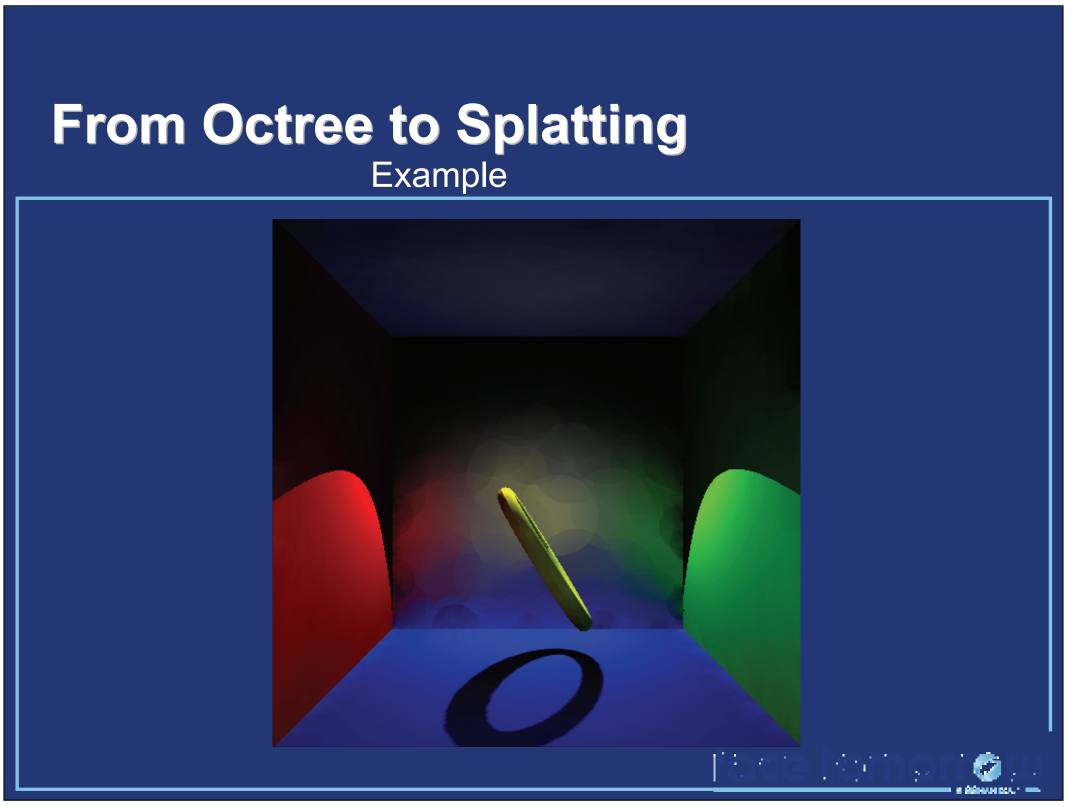


From Octree to Splatting

Final Image Generation

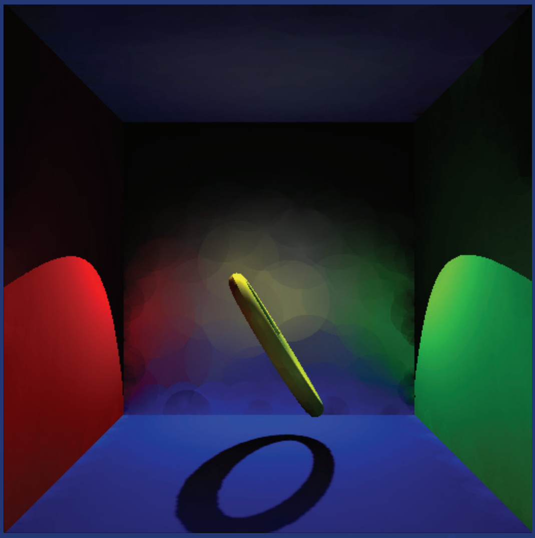
$$\frac{\sum_k w_k(P)E(P)}{\sum_k w_k(P)}$$

Once all the necessary records have been splatted, the result of the irradiance interpolation equation can then be obtained by dividing the weighted sum of contributions by the sum of the weights. This operation can be easily performed within a fragment shader, by dividing the RGB components (that is, the weighted sum of contributions) by the alpha channel (the sum of weights).



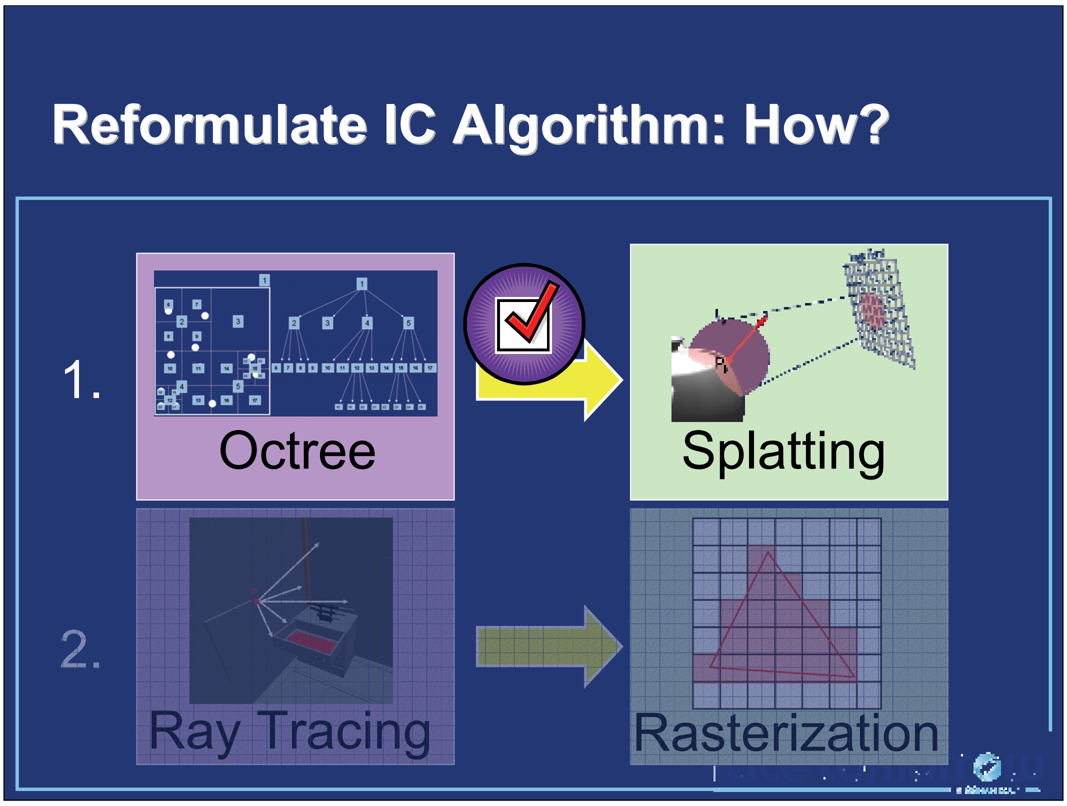
From Octree to Splatting

Example



To summarize, let us consider an example. At the beginning of the algorithm, the cache contains no records. Hence the generated image only features direct illumination. Then, we add a record on the back wall. This record contributes to the points within its neighborhood. When adding a second record, the zones of influence overlap. For the pixels within the overlapping area, the estimated irradiance is calculated using the weighted average of the contributions. Other records are successively added to the cache, until the entire visible area of the scene is covered by the zones of influence of the records. The resulting image features both direct and indirect lighting for every visible point.

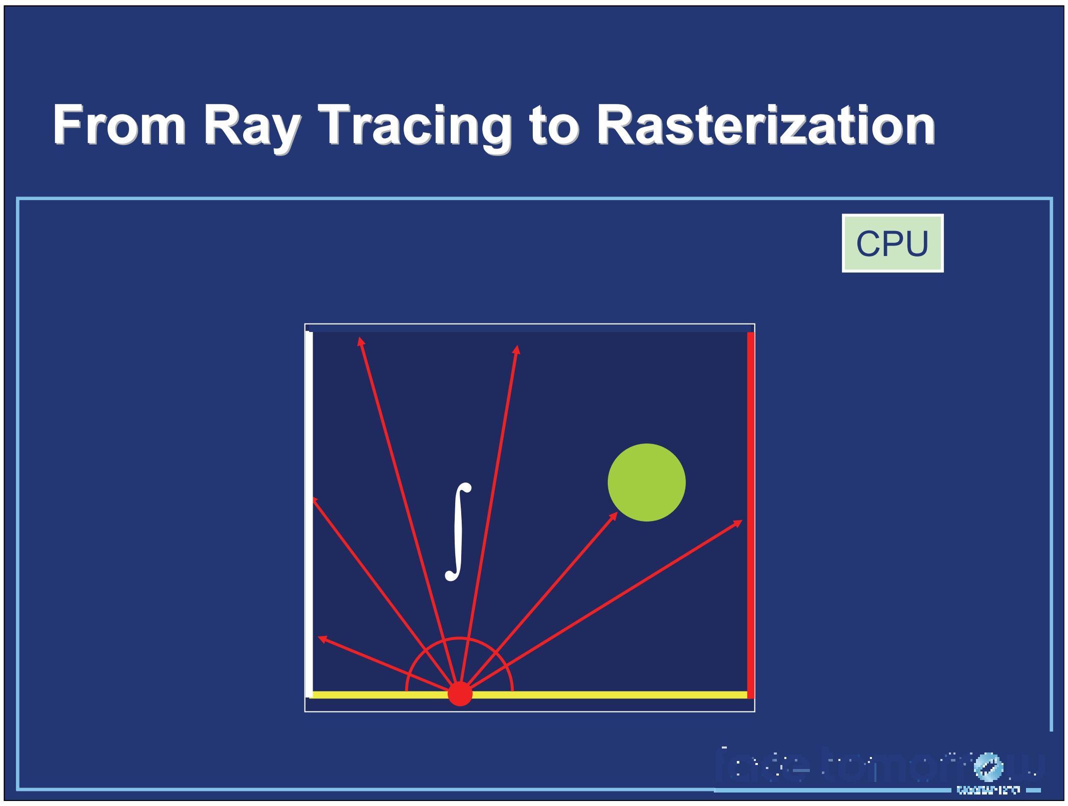
Note that for explanation purposes, the reconstruction of the indirect lighting is very coarse to highlight the zones of influence of each record. Also, the gradients are not used. As with the classical irradiance caching algorithm, high quality can be obtained by using irradiance gradients and setting an appropriate value for the user-defined parameter a.



Reformulate IC Algorithm: How?

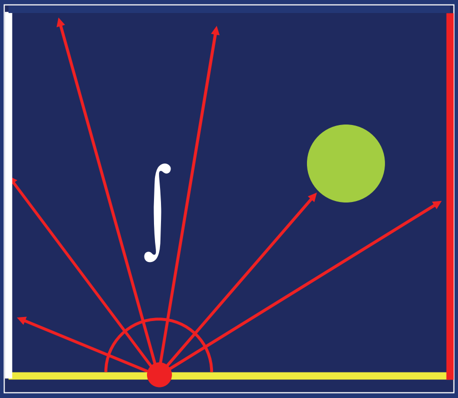
1. Octree → Splatting

2. Ray Tracing → Rasterization

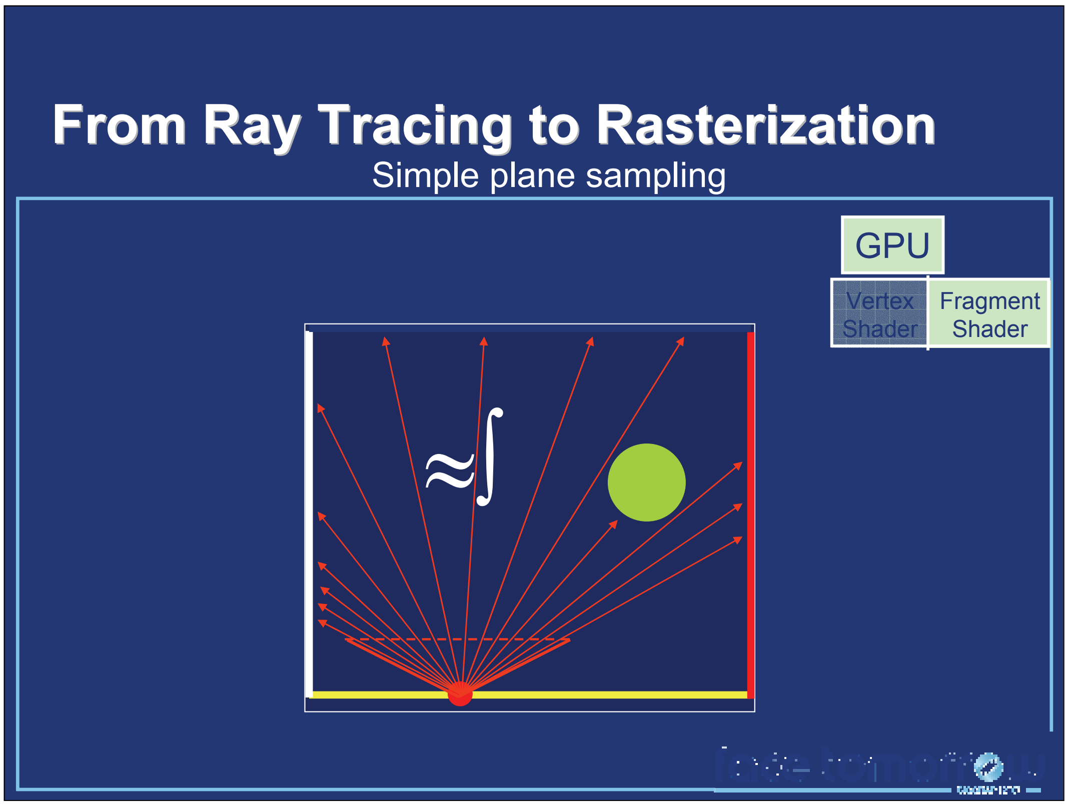


From Ray Tracing to Rasterization

CPU



In classical irradiance caching, the irradiance at a point is estimated by Monte Carlo ray tracing: random rays are traced through the scene, gathering the lighting incoming from the surrounding environment. This estimate is usually computed on the CPU.



From Ray Tracing to Rasterization

Simple plane sampling

A well-known approximate method for hemisphere sampling on the GPU is the simple plane sampling: a virtual camera with a large aperture is placed at the point of interest. The scene is then rendered on graphics hardware, yielding an image of the surrounding objects. On recent graphics hardware, this data can be computed in high dynamic range (HDR) using floating-point render target and programmable shaders. The shadowing effects can be efficiently accounted for using fast shadowing techniques such as shadow mapping [Wil78].

The solid angle subtended by a pixel p is defined as $\Omega_p = A*\cos(\theta)/(d*d)$ where:

•A is the surface of a pixel

•θ is the angle between the direction passing through the pixel and the surface normal

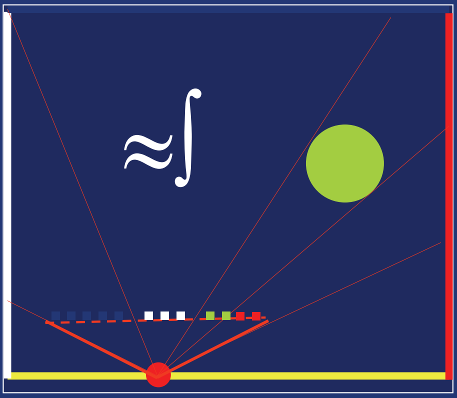
•d is the distance between the point of interest and the image plane

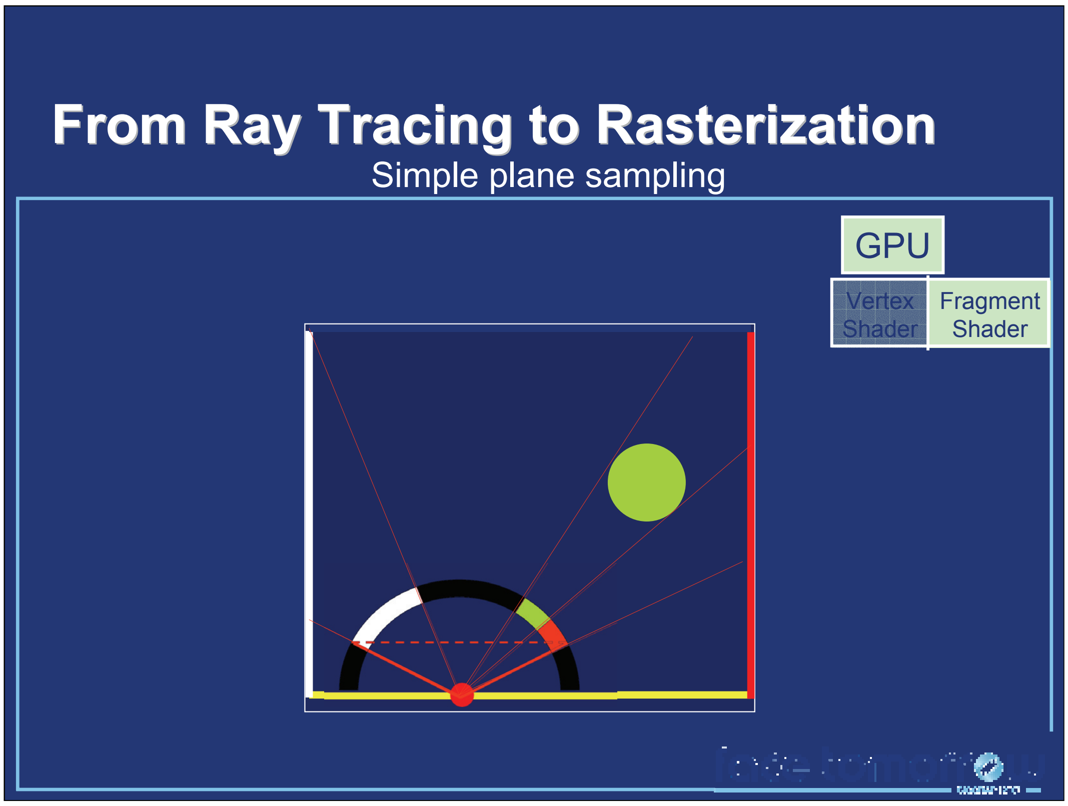
From Ray Tracing to Rasterization

Simple plane sampling

GPU

Vertex Shader | Fragment Shader





From Ray Tracing to Rasterization

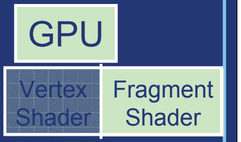
Simple plane sampling

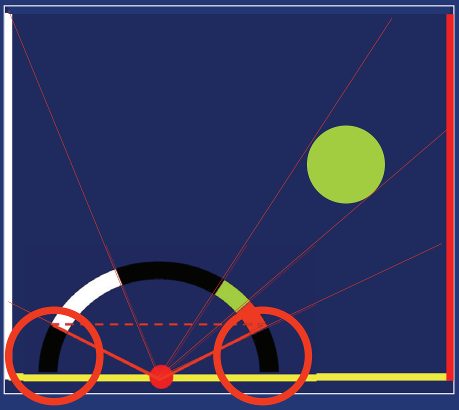
However, the aperture of the camera cannot allow us to sample the entire hemisphere: a perspective camera is represented by a perspective projection matrix which is applied to the visible contents of the scene. In OpenGL, such a camera is modeled using the gluPerspective function, whose values are calculated using the field of view (FOV) of the camera. More precisely, a key value in this matrix is f = cot(FOV/2), which is undefined for FOV=180°. Furthermore, using a very large aperture such as 179.999° leads to important perspective deformation and sampling problems.

In [LC04], Larsen et al. show that an aperture of 126.87° is sufficient for capturing 86% of the incoming directions. However, the remaining 14% are unknown and must be compensated to avoid a systematic underestimation of the incoming radiance. Furthermore, this compensation must respect the directional information of the incoming radiance to allow for a later implementation of radiance caching for global illumination computation on glossy surfaces.

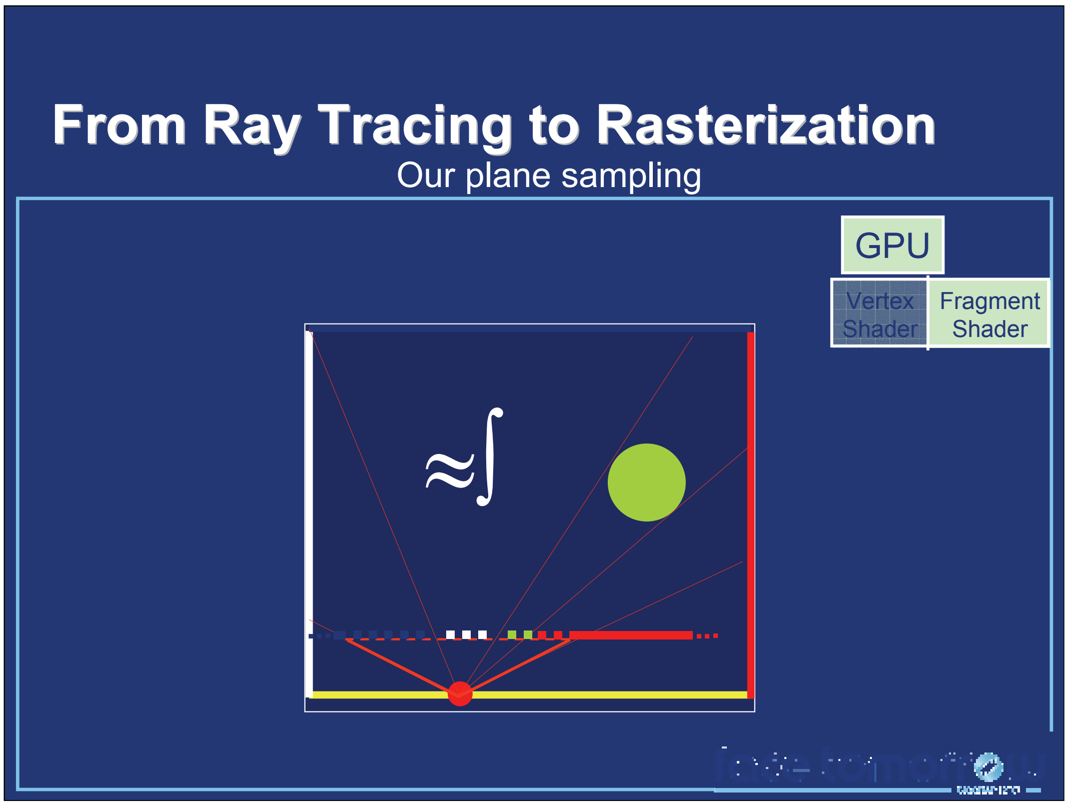
From Ray Tracing to Rasterization

Simple plane sampling





Incoming radiance loss



From Ray Tracing to Rasterization

Our plane sampling

We propose a very simple compensation method, in which the border pixels are virtually « extended » to fill the parts of the hemisphere which have not been actually sampled. To this end, border pixels are considered as covering a solid angle of:

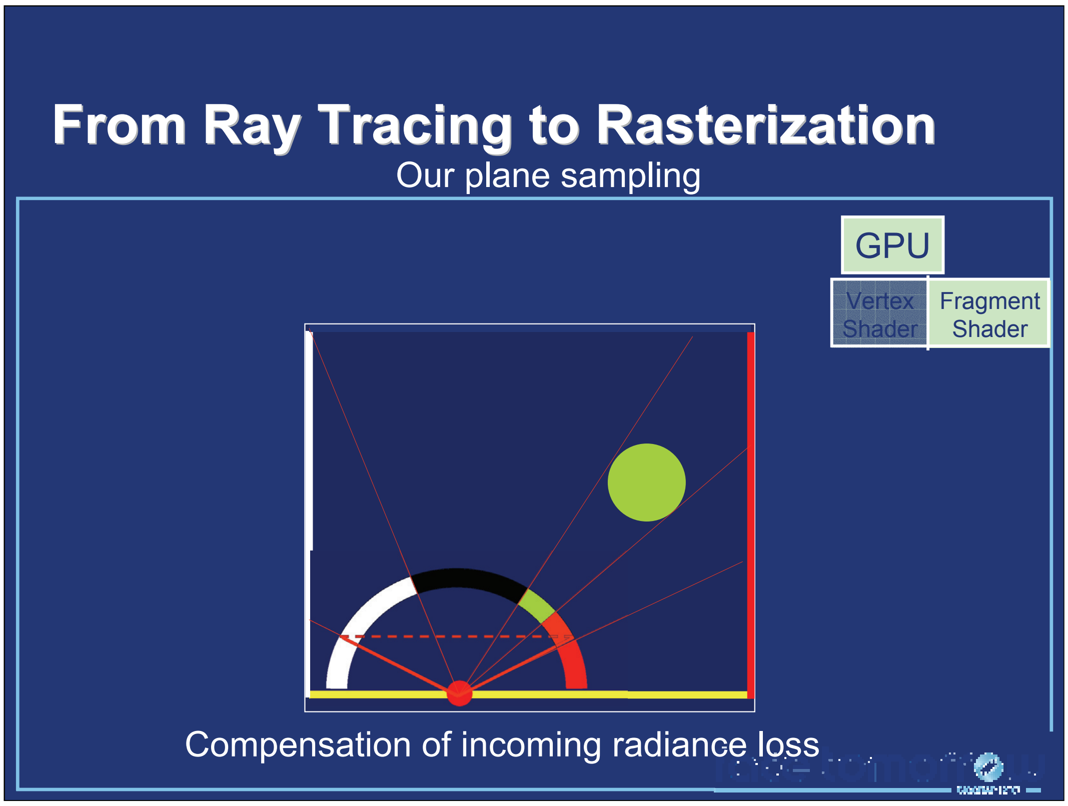
$\Omega_{border} = \Omega_p + \cos(\theta_{border}) * \delta_\Phi$

Where:

•Ω_p is the solid angle subtended by the pixel

•θ_{border} is the largest θ covered by the aperture of the camera

•δ_Φ is the interval of Φ spanned by the pixel



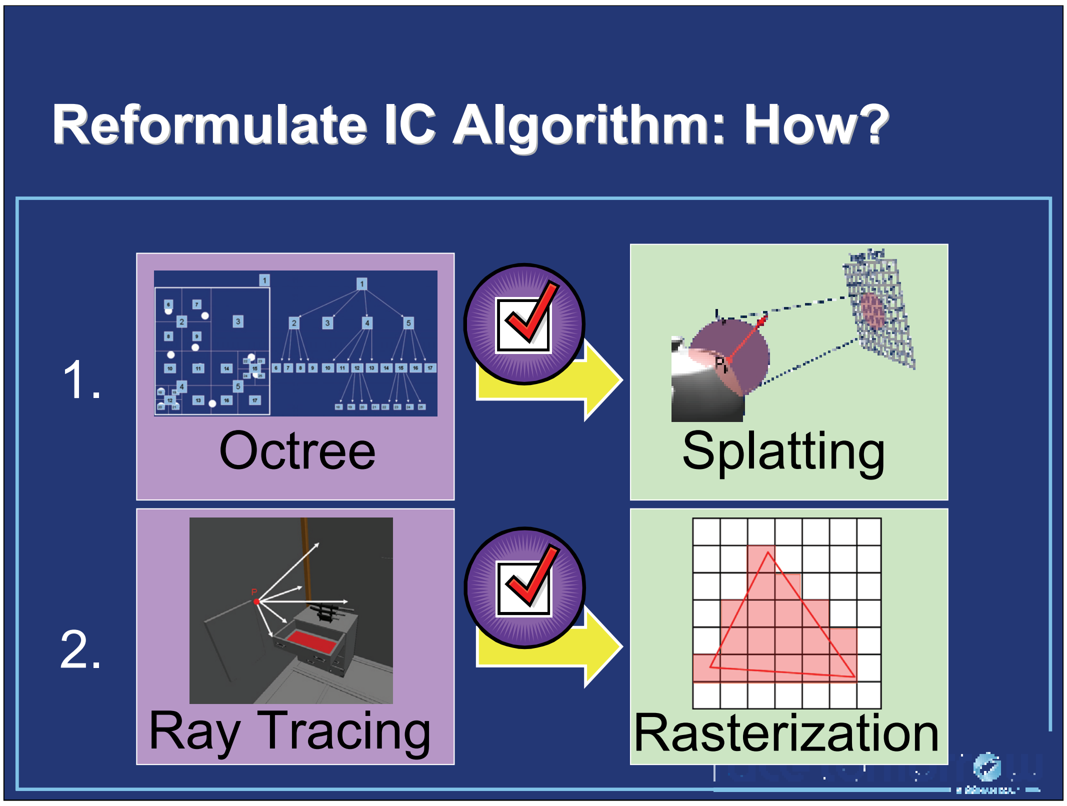
From Ray Tracing to Rasterization

Our plane sampling

Compensation of incoming radiance loss

This allows us to compensate the missing information by extrapolating the radiance values at the extremities of the sampling plane. This extrapolation provides a plausible estimate of the missing radiances, hence making it suitable for radiance caching also.

It must be noted that other techniques can be used, such as the full hemicube sampling (which requires several passes), or a hemispherical parametrization of the hemisphere in vertex shaders.

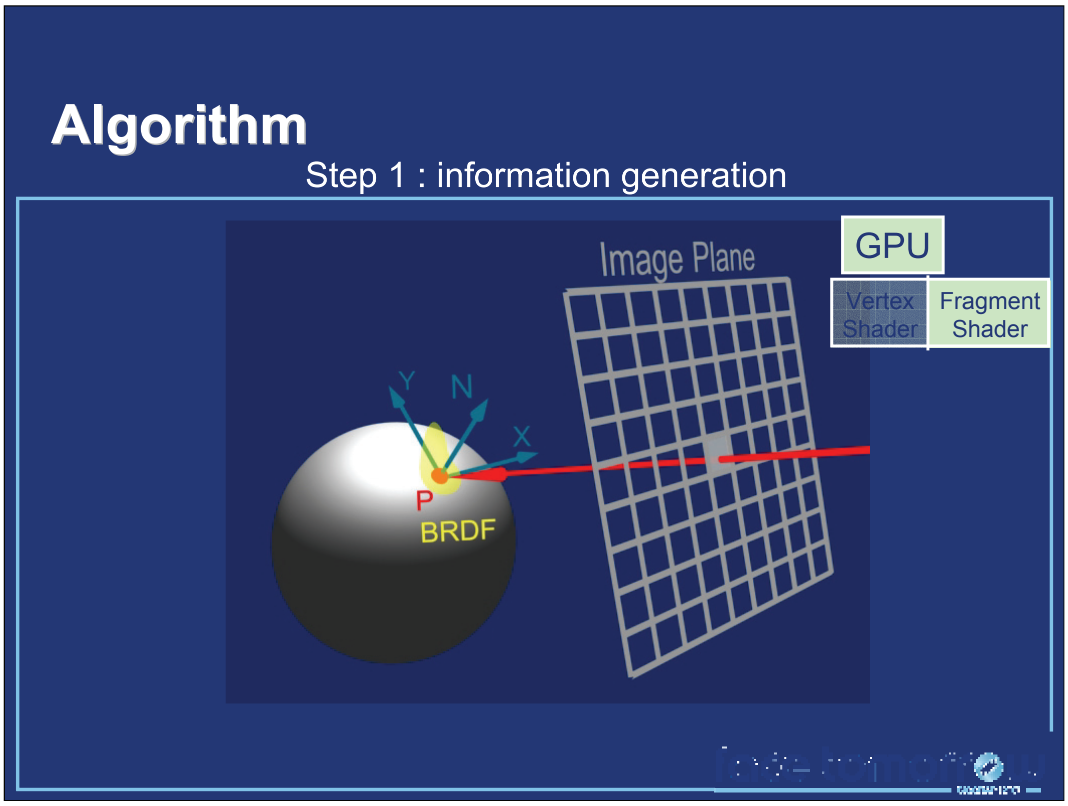


Reformulate IC Algorithm: How?

1. Octree → Splatting

2. Ray Tracing → Rasterization

Now the algorithm has been reformulated for implementation on graphics hardware. Next section will present the entire algorithm for global illumination computation using irradiance splatting.

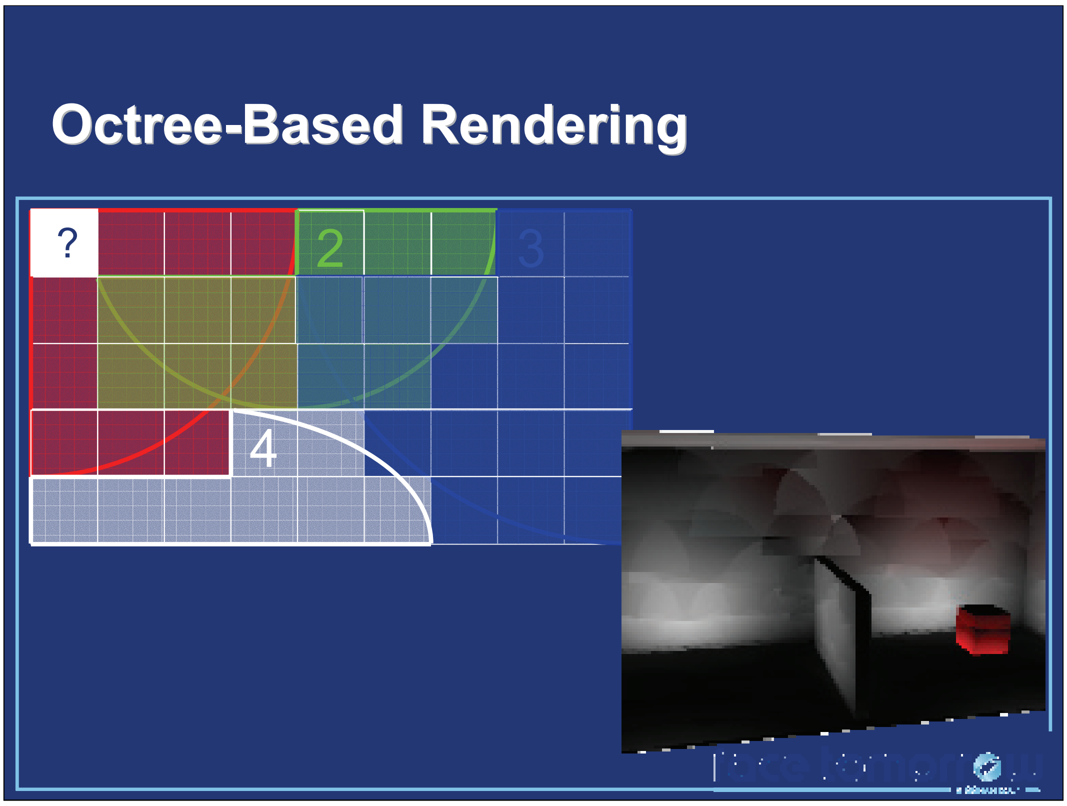


Algorithm

Step 1 : information generation

The first step of the algorithm consists in obtaining basic information about the points visible from the user point of view. In particular, this information includes the position and normal of the visible points. This data can be easily generated in one pass on the GPU using floating-point multiple render targets and programmable shaders.

In the next step, the algorithm transfers this data into the main memory. The CPU is then used to traverse the image and detect where new records are required to render the image.

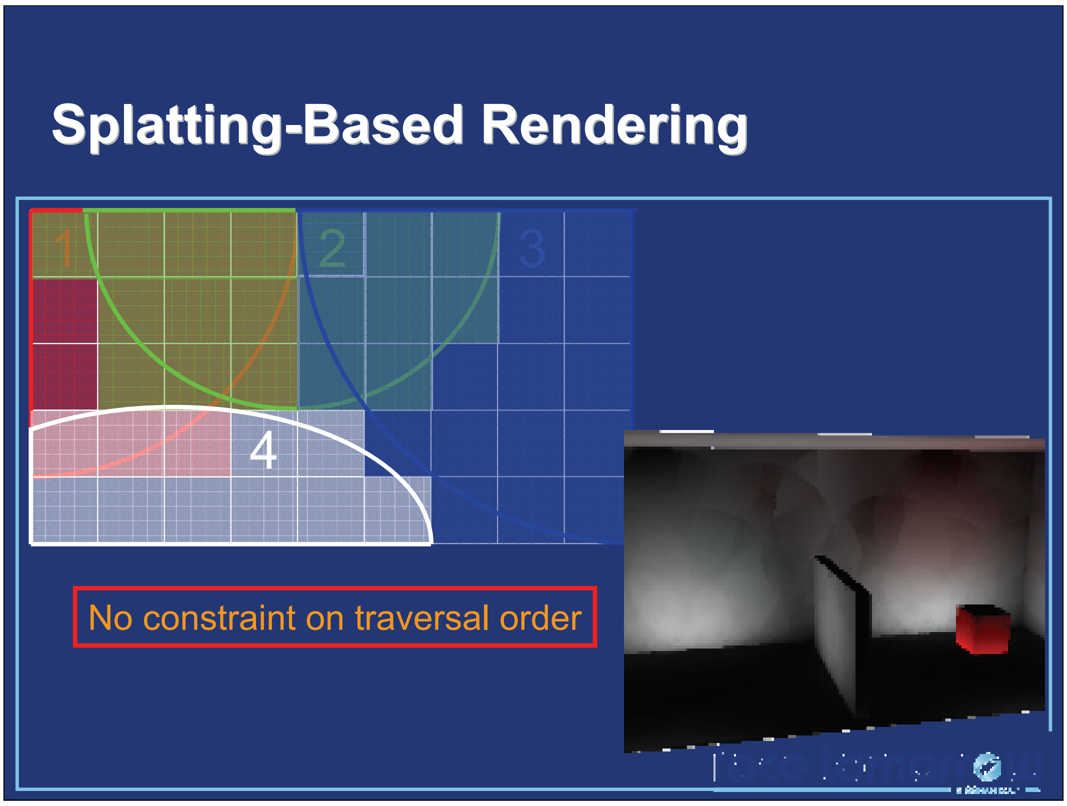


Octree-Based Rendering

Usually, the detection is performed by querying the irradiance cache (hence the octree) for each pixel as follows:

```
•For each visible point P with normal N corresponding to pixel p
        •W = GetSumOfContributions(P,N)
        •If (W < a)
                •R = CreateNewRecord(P,N)
                •IrradianceCache.StoreRecord(R)
                •p.radiance = R.irradiance*SurfaceReflectance(P)
        •Else
                •E = EstimateIrradiance(P,N)
                •P.radiance = E*SurfaceReflectance(P)
        •EndIf
•EndFor
```

While this algorithm ensures the presence of a sufficient number of records, such records are not used in an optimal way: when record 2 is created, the algorithm only propagates the contribution of 2 to the pixels which have not been checked yet. The previous pixels remain unchanged, even though record 2 may contribute to their radiance. If the image is traversed linearly, disturbing artifacts may appear: in this example the image is traversed from bottom to top. Therefore, the records contribute only to the points located above them in the image. This problem is usually compensated by using other traversal algorithms, typically based on a hierarchical subdivision of the image.



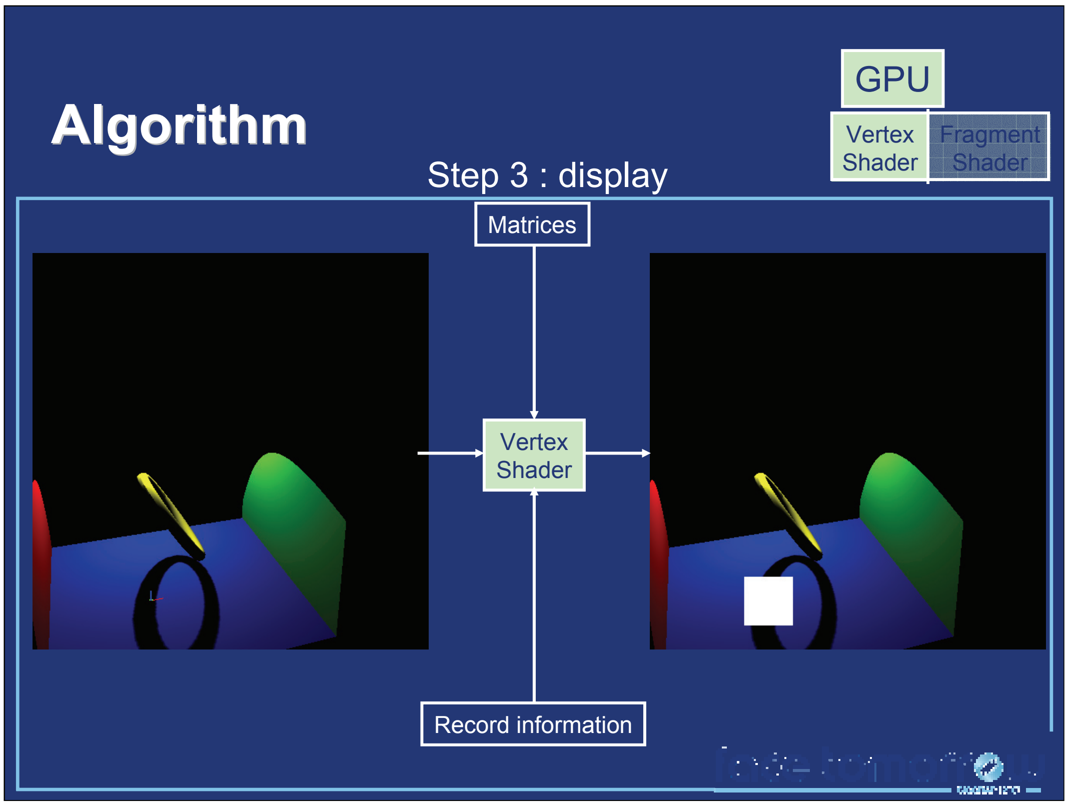
Splatting-Based Rendering

No constraint on traversal order

When using irradiance splatting, the detection code becomes:

```
•For each visible point P with normal N corresponding to pixel p
    •W = GetSumOfContributions(P,N)
    •If (W < a)
        •R = CreateNewRecord(P,N)
        •IrradianceCache.StoreRecord(R)
        •SplatRecord(R)
        •p.radiance = R.irradiance*SurfaceReflectance(P)
    •Else
        •// Nothing to do: the radiance value depends of surrounding records and may
        be updated
    •EndIf
•EndFor
```

In our method, the records are splatted onto their entire zone of influence: even pixels which have been previously checked can be updated by the addition of a novel record. Hence this method removes the constraint on the image traversal order. In the example image, we used a simple linear traversal. The zone of influence of each record is completely accounted for by our splatting method.



Once all records have been computed, each record must be rendered on the GPU. Let us consider a record located at point P with normal N, and with an harmonic mean distance to surrounding objects H. The zone of influence of the record is contained within a screen-aligned square. Let us consider the vertices of a square such that:

v0 = (1.0, 0.0, 0.0)

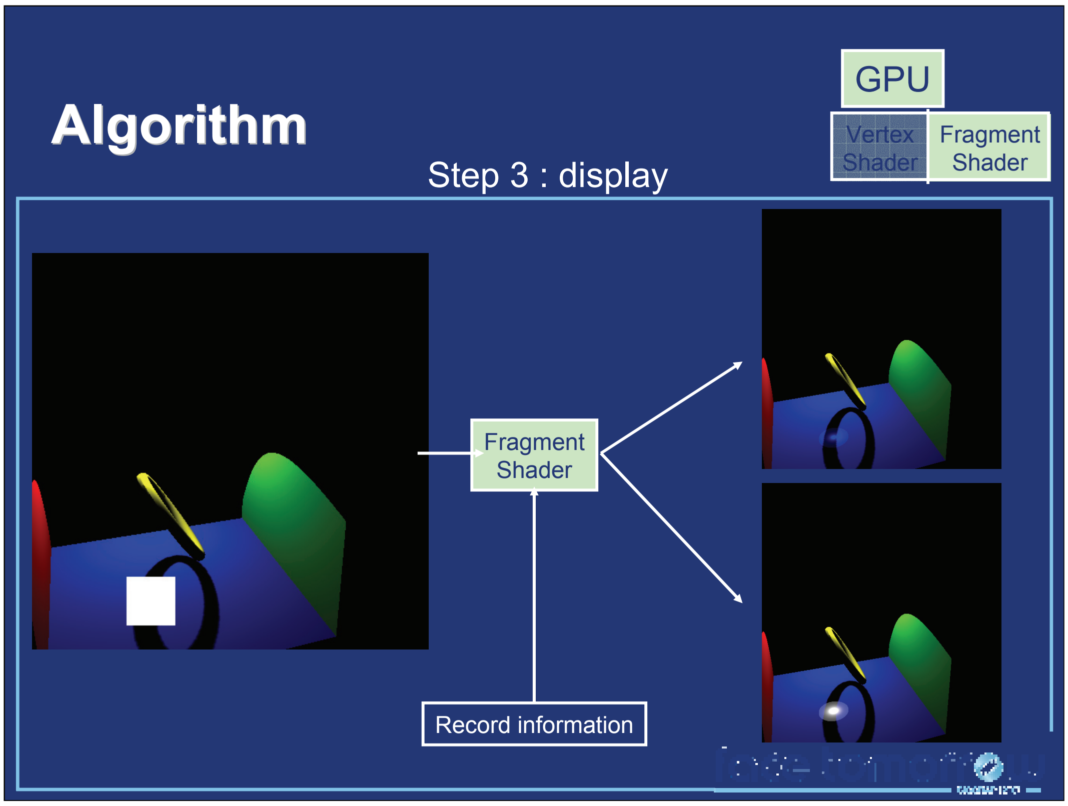
v1 = (1.0, 1.0, 0.0)

v2 = (0.0, 1.0, 0.0)

v3 = (0.0, 0.0, 0.0)

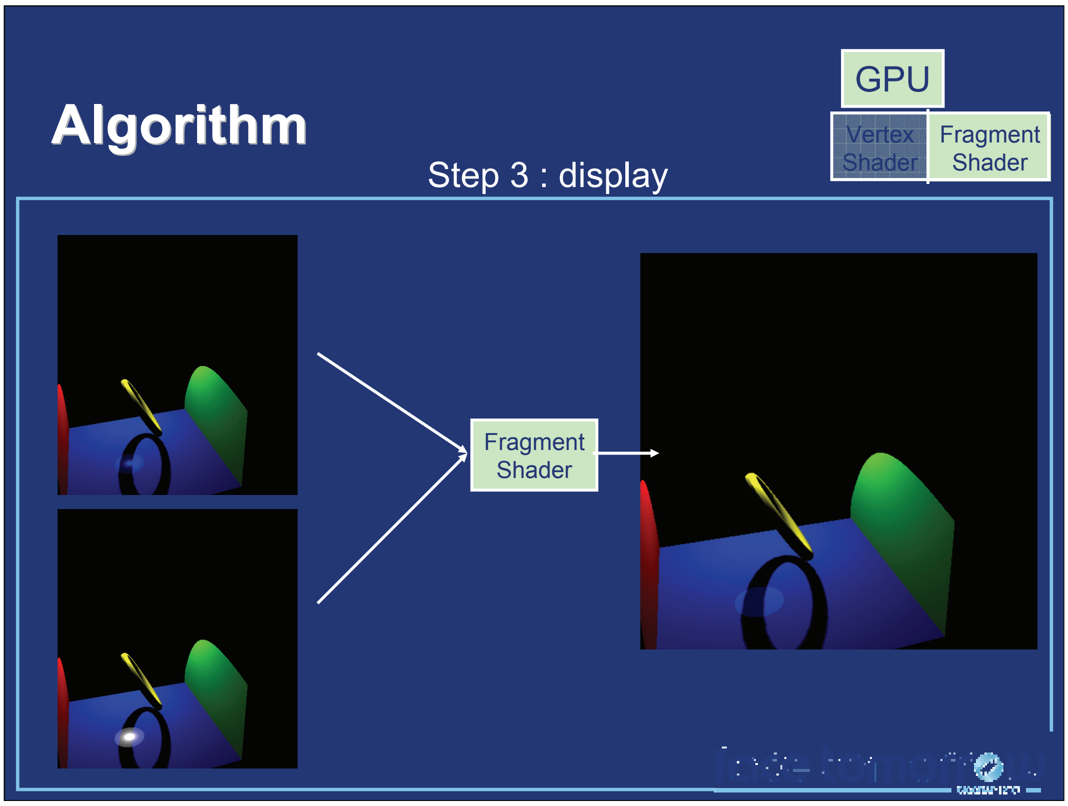
Each vertex can be transformed into screen space using the following function:

```
float4 TransformVertex(Vertex v)
{

            float4 projPos = ModelViewMatrix*P; // Transform the point into camera space
            float scale = projPos.w;
            float4 projPos /= projPos.w;

            float radius = a*H; // Radius of the zone of influence in camera space
```



The fragment shader then computes the actual value of the weighting function for each point within the quadrilateral, using the formula defined in [Ward88]. If the record is allowed to contribute to the lighting of a visible point, its contribution is computed using irradiance gradients such as in [Ward92]. The output of the shader for the corresponding fragment is a HDR RGBA value, where RGB represents the weighted contribution of the record, and A represents the weight of the record's contribution.

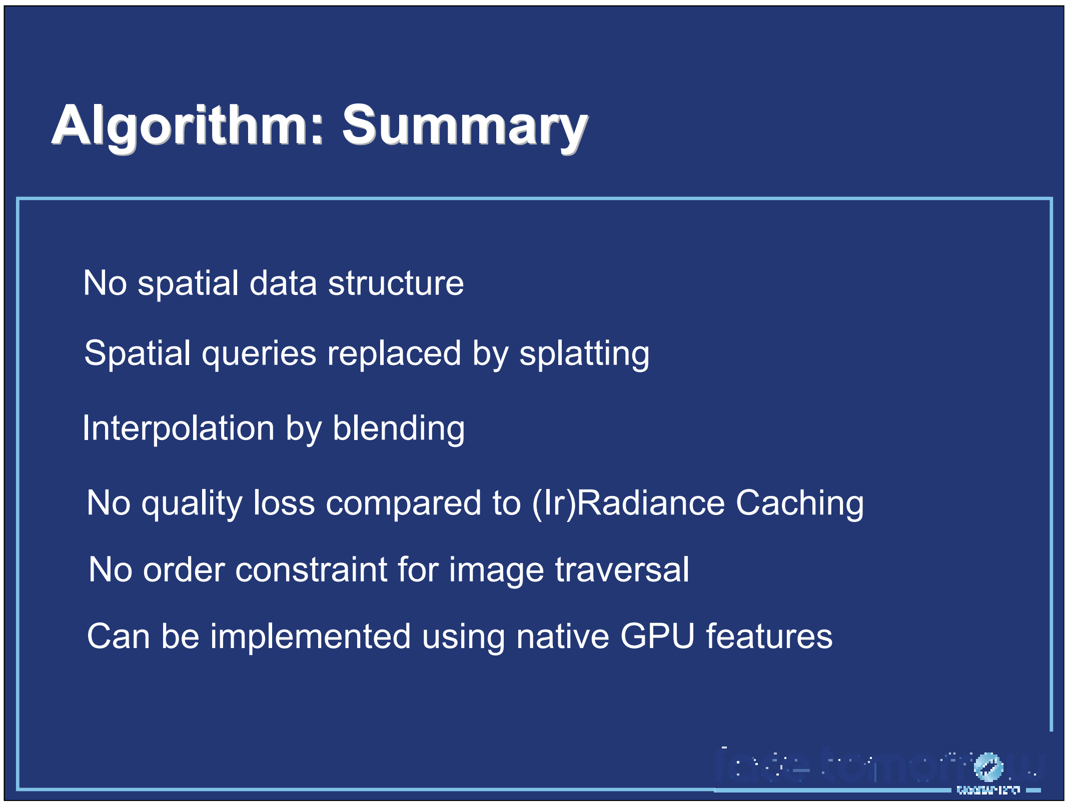
When several records are rendered, their RGBA values are simply added together using the floating-point alpha blending of graphics hardware (glBlendFunc(GL_ONE, GL_ONE)). This yields a sum of weighted contributions in the RGB components, and a sum of weights in the A component.



Algorithm

Step 3 : display

In a final step, a texture containing the RGBA values discussed above is used in a final render pass. This pass renders a single screen-sized quadrilateral. For a given pixel the fragment shader fetches the corresponding RGBA value, and outputs RGB/A to perform the irradiance estimation described in [Ward88].



Algorithm: Summary

No spatial data structure

Spatial queries replaced by splatting

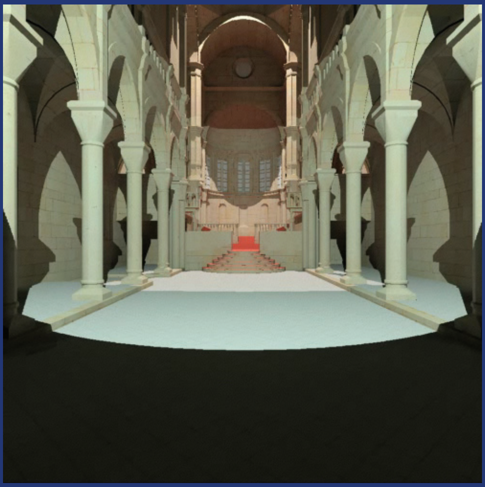
Interpolation by blending

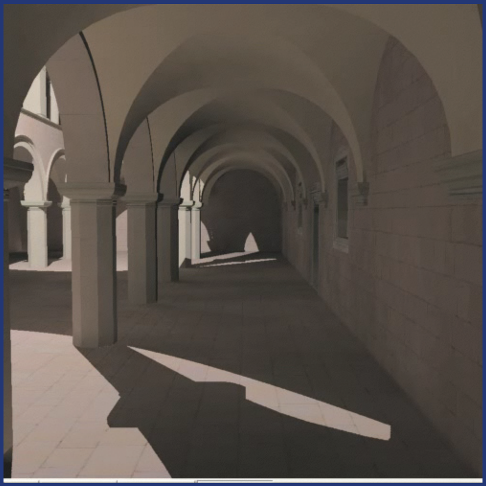
No quality loss compared to (Ir)Radiance Caching

No order constraint for image traversal

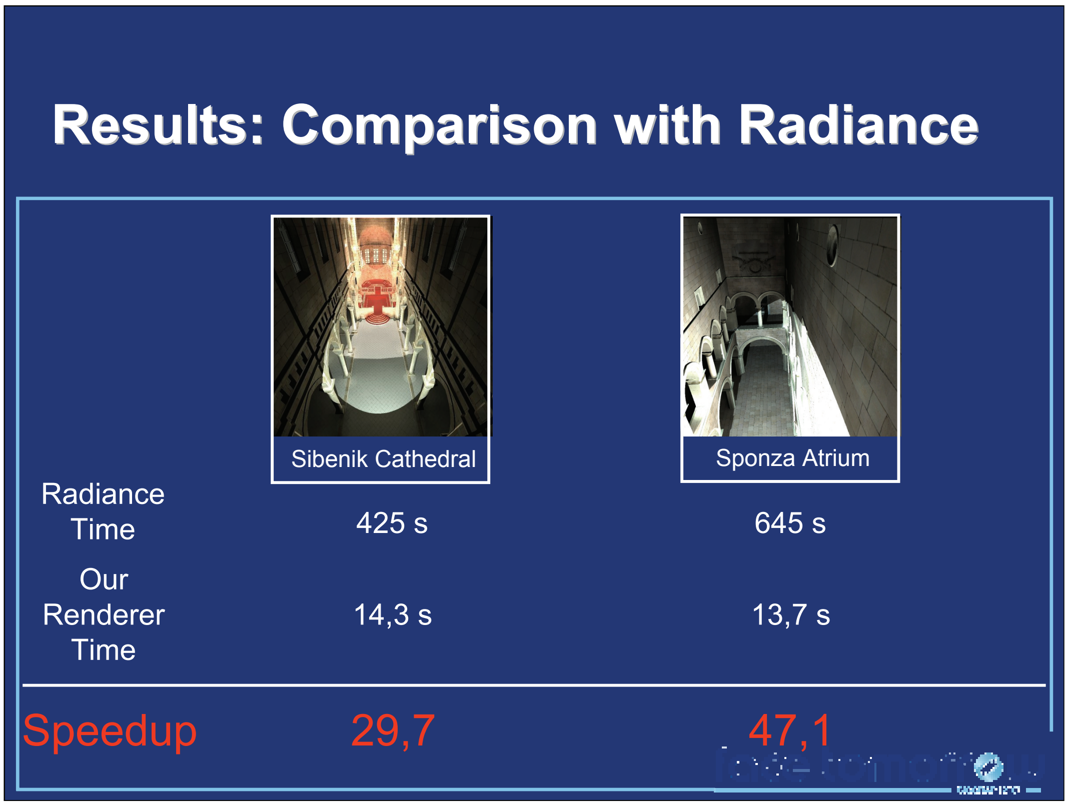
Can be implemented using native GPU features

Results

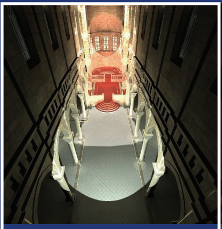

Sibenik Cathedral (80k tri.)


Sponza Atrium (66K tri.)

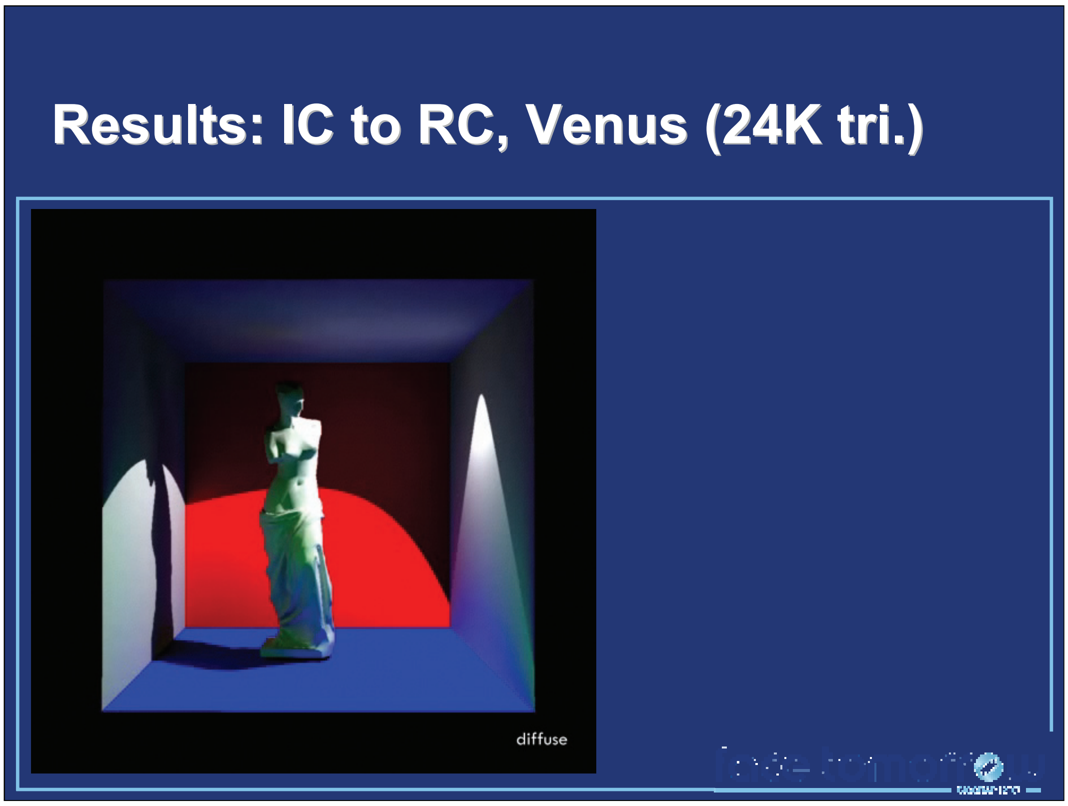
Those videos were rendered on a GeForce 6800. The scene being static, the irradiance cache is reused across frames. Hence the first frame has been computed in approximately 20s, while the other frames took approximately 1s to render.



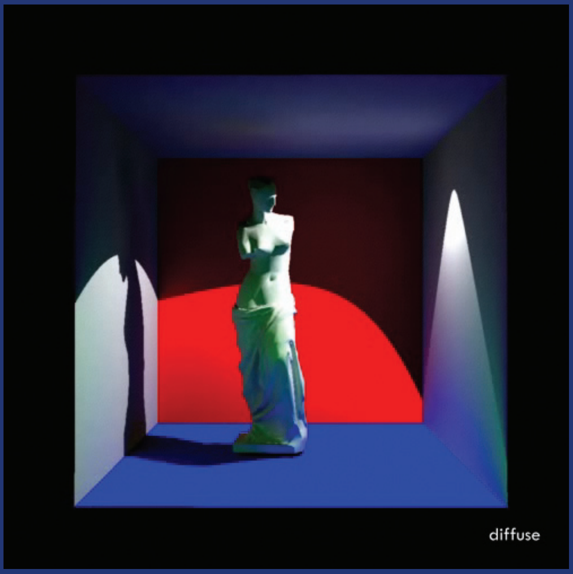
Results: Comparison with Radiance

	 Sibenik Cathedral	Sponza Atrium
Radiance Time	425 s	645 s
Our Renderer Time	14,3 s	13,7 s
Speedup	29,7	47,1

A comparison between our GPU-based method and the Radiance Software.



Results: IC to RC, Venus (24K tri.)



This method can be straightforwardly extended to glossy global illumination using radiance caching.

Temporal Coherence

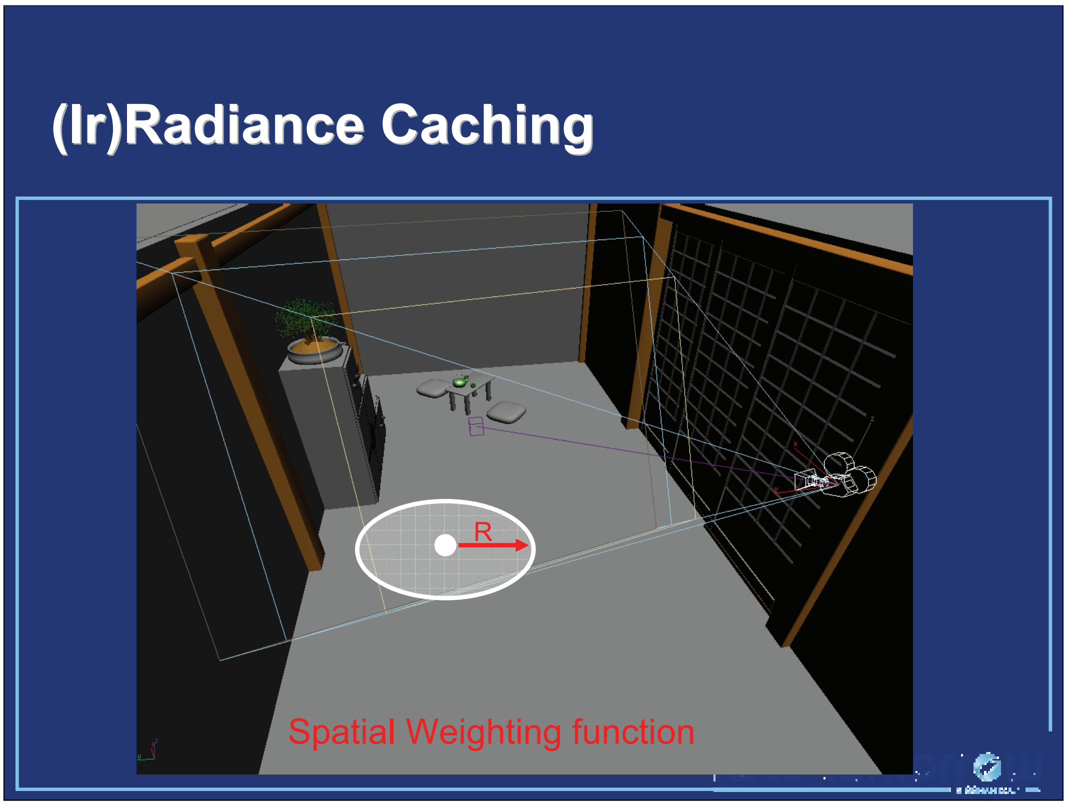
SIGGRAPH2007

Pascal Gautron

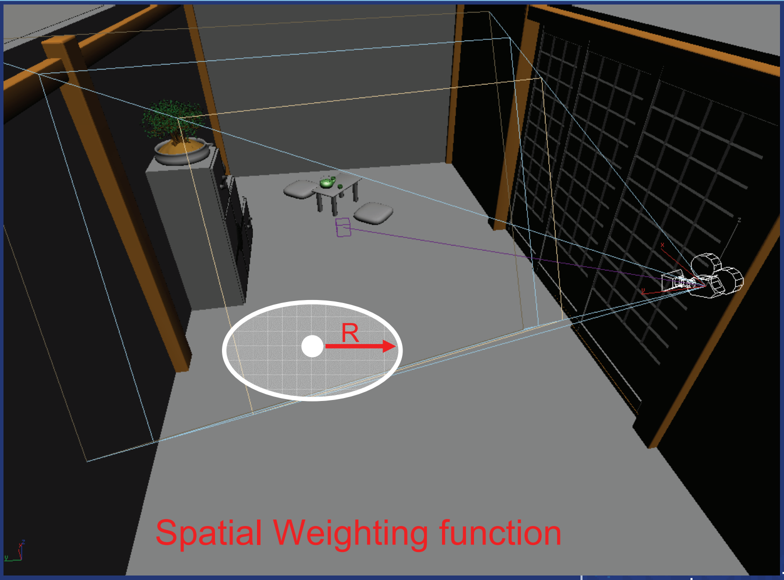
Post-Doctoral Researcher

France Telecom R&D Rennes

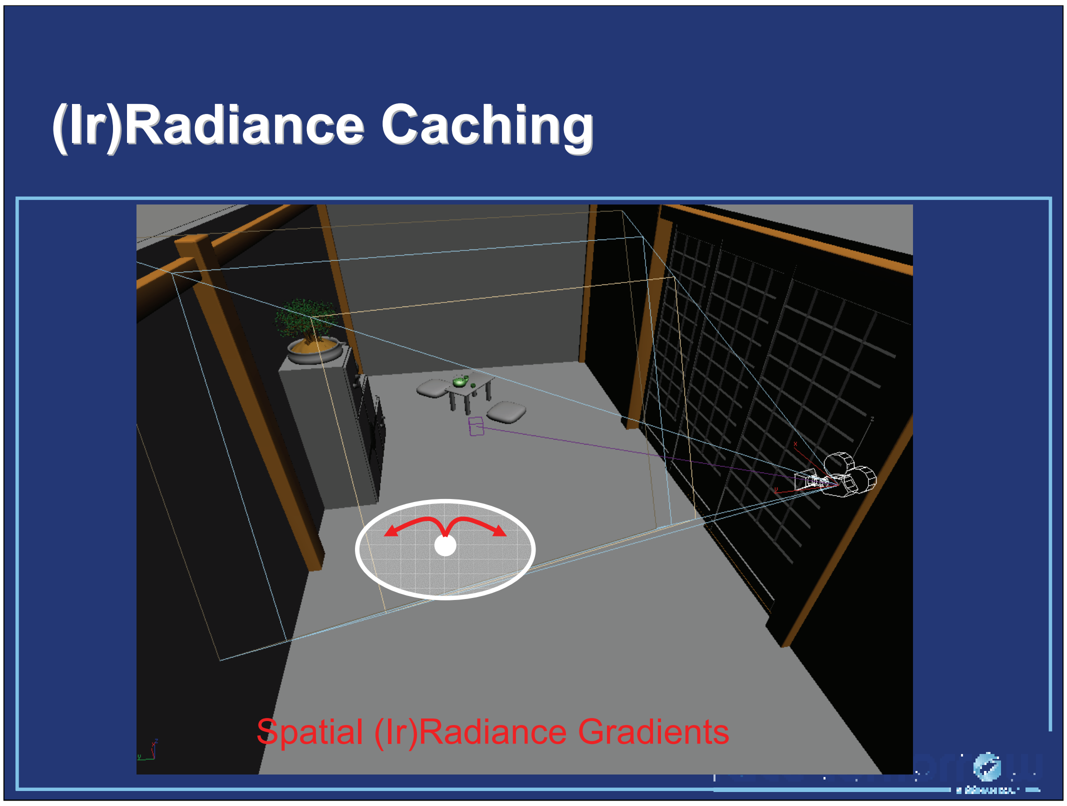
France



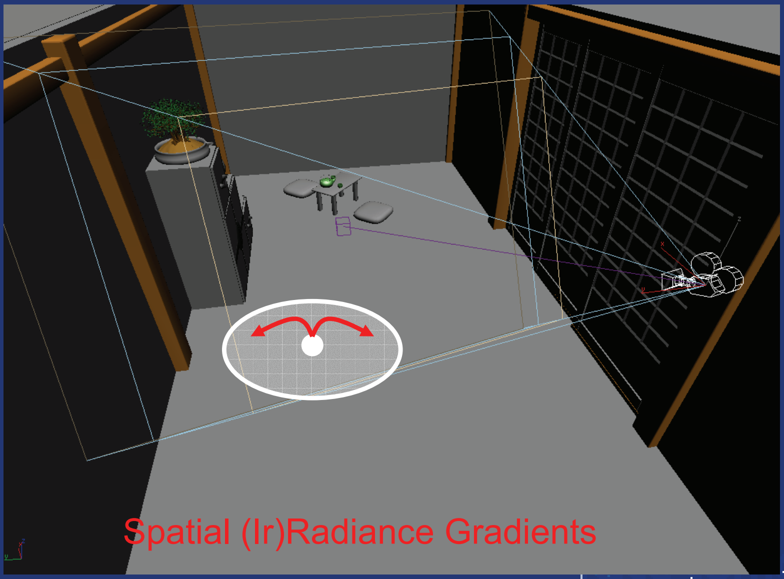
(Ir)Radiance Caching



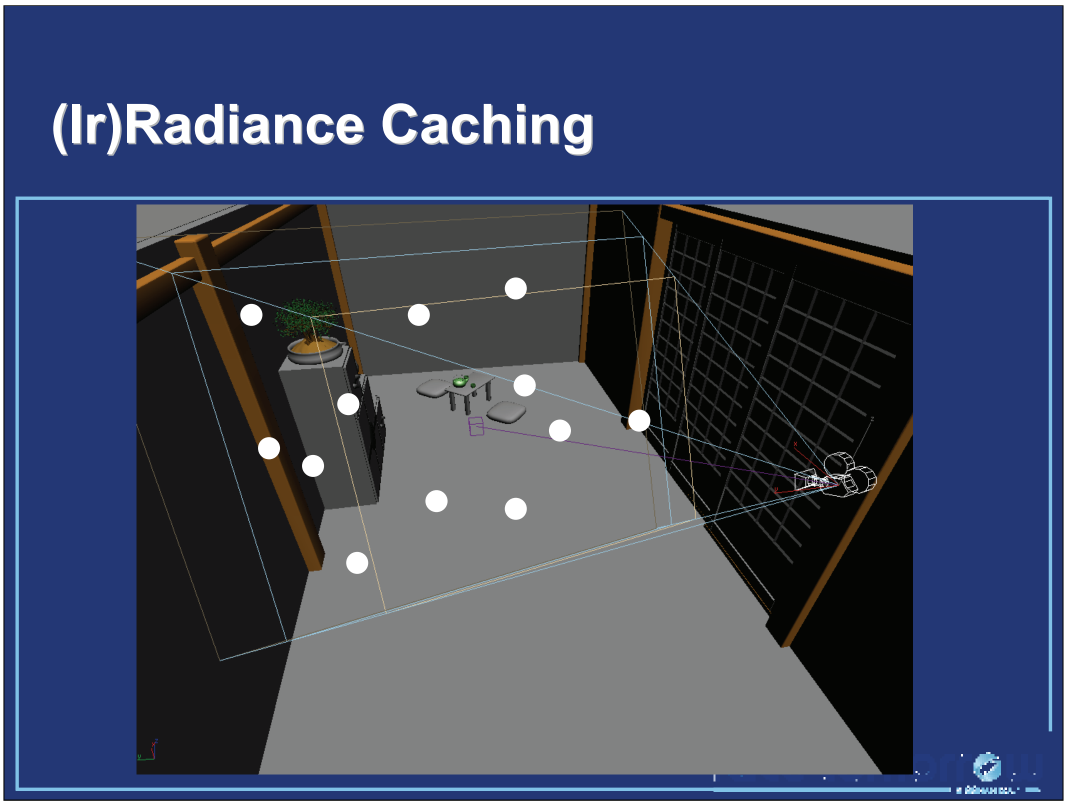
As explained in the previous parts of this course, the zone of influence of a given irradiance record is defined by a spatial weighting function.



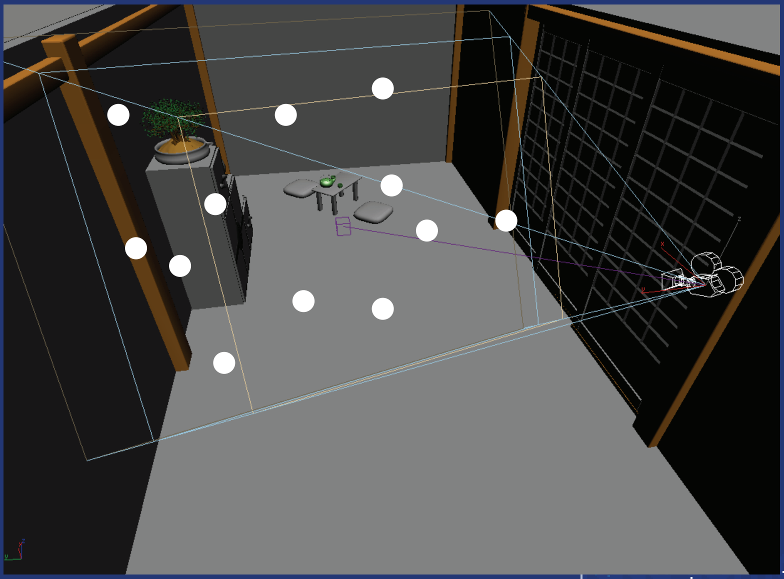
(Ir)Radiance Caching



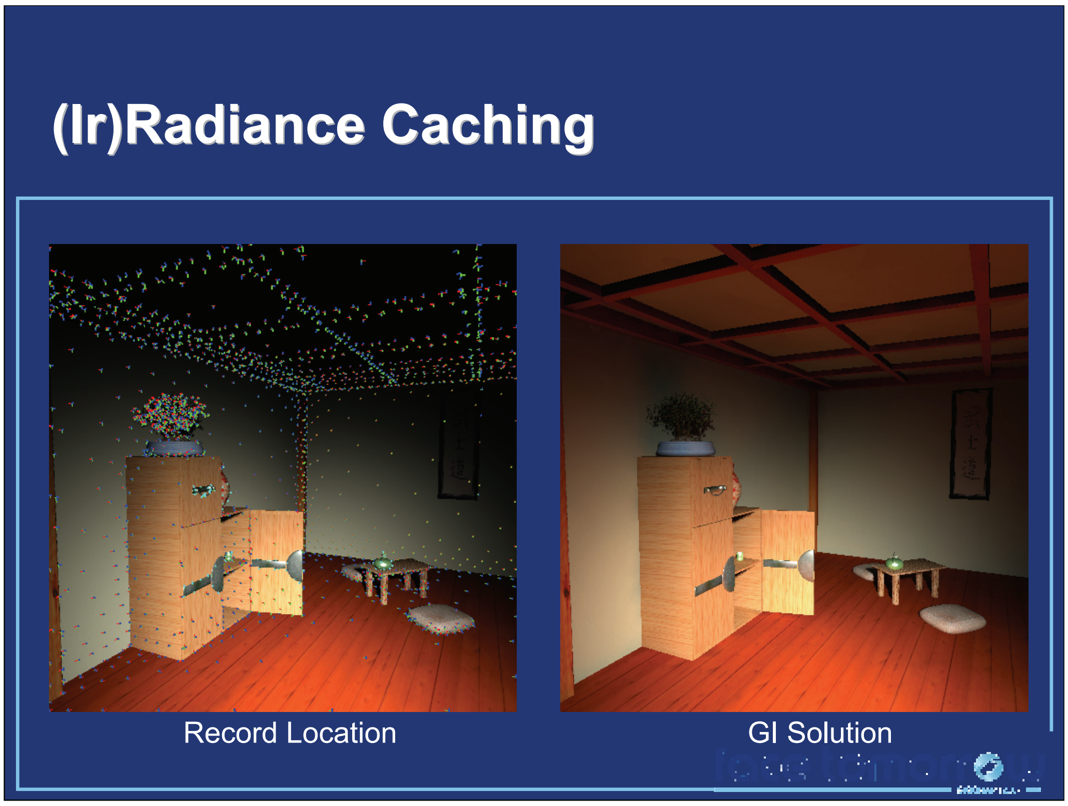
The contribution of a record within its zone of influence is estimated using irradiance gradients [Ward92].



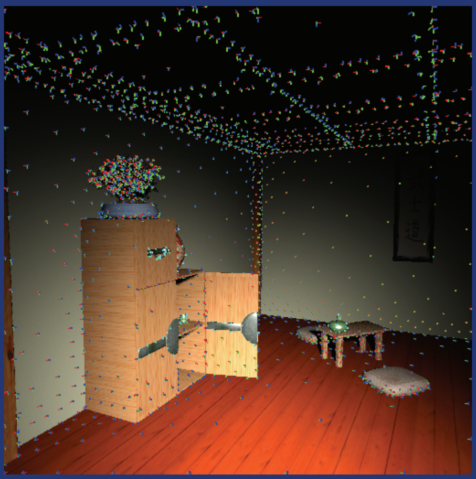
(Ir)Radiance Caching

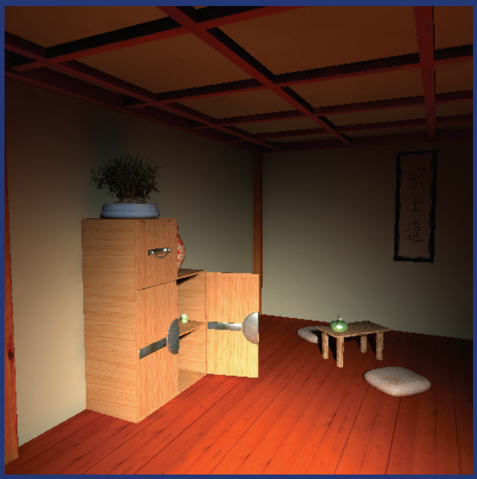


The lighting of the visible points is then computed explicitly at the location of the records, and **extrapolated** for all the other visible points.

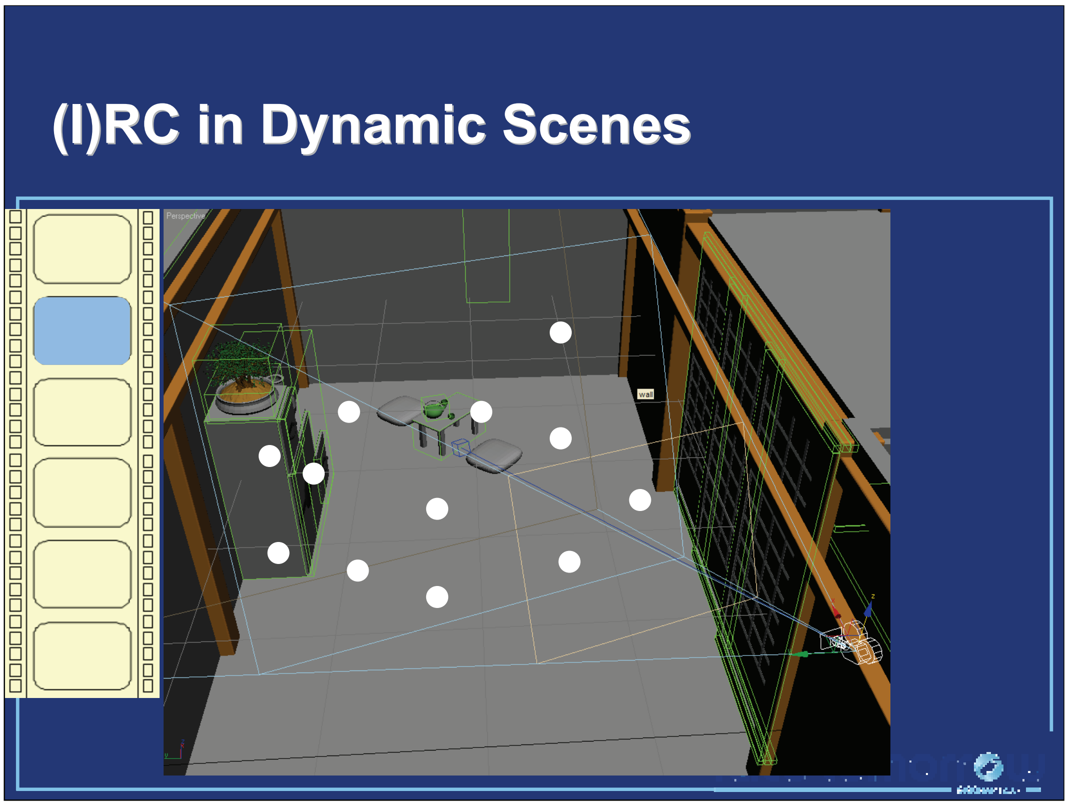


(Ir)Radiance Caching

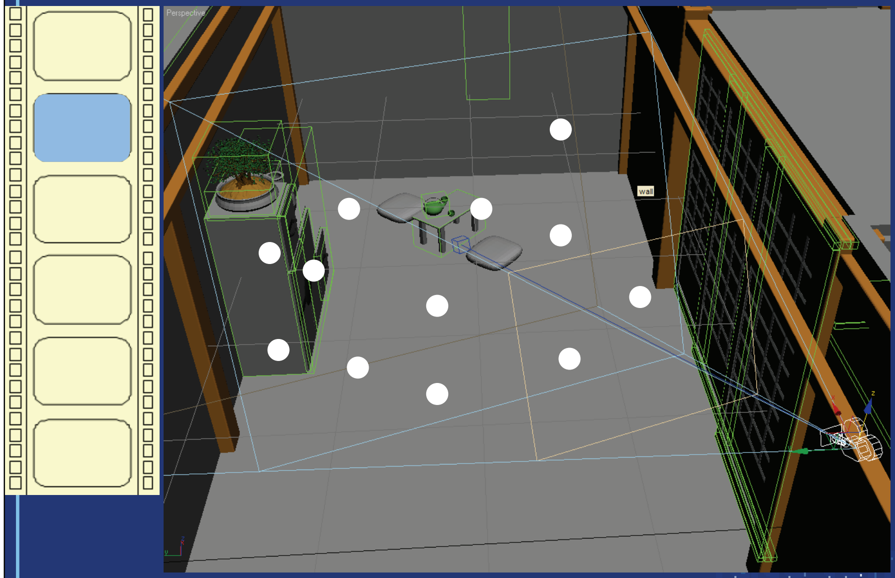

Record Location


GI Solution

In a single image, this method provides high quality results even when using a very sparse sampling. However, since the estimated lighting is generally obtained through extrapolation from the location of the records, the result depends on the **distribution** of the records.



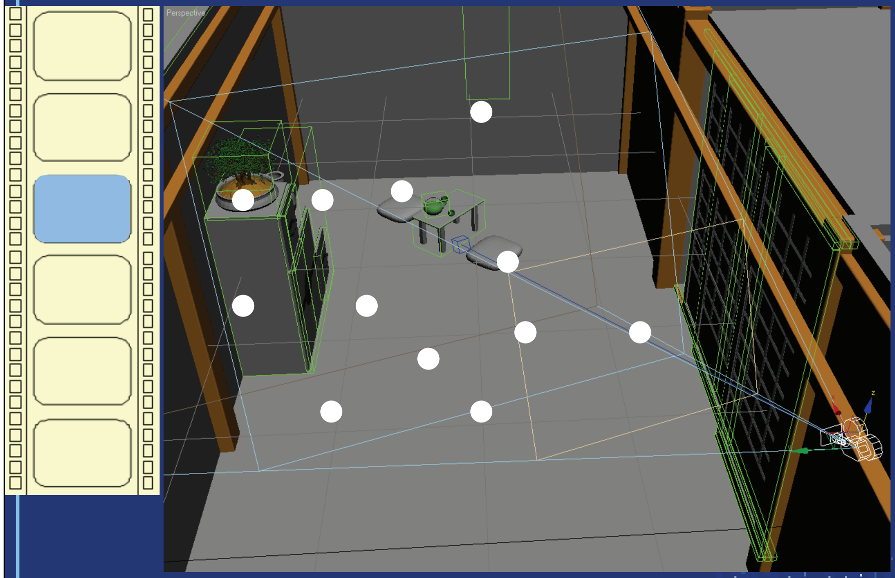
(I)RC in Dynamic Scenes



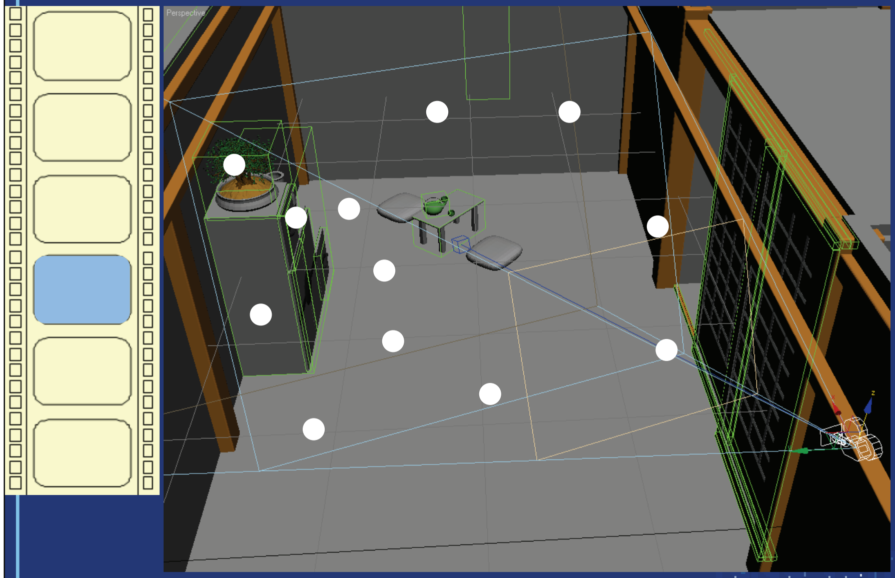
In dynamic scenes, the indirect lighting must be changes in each frame. A simple method for animation rendering using irradiance caching is the entire recomputation of the global illumination solution for each frame. The camera and objects being dynamic, the distribution of records is likely to change across frames, yielding flickering artifacts. A video illustrating this method can be found on [MyWebSite].

Furthermore, the indirect lighting tends to change slowly across frames. The indirect lighting computed at a given frame may thus be reused in several subsequent frames without degrading the quality. However, the lighting may change very quickly in some areas of the scene, and be nearly-static elsewhere. The method described in this course leverages these observations by assigning a distinct lifespan to each record in the cache.

(I)RC in Dynamic Scenes



(I)RC in Dynamic Scenes

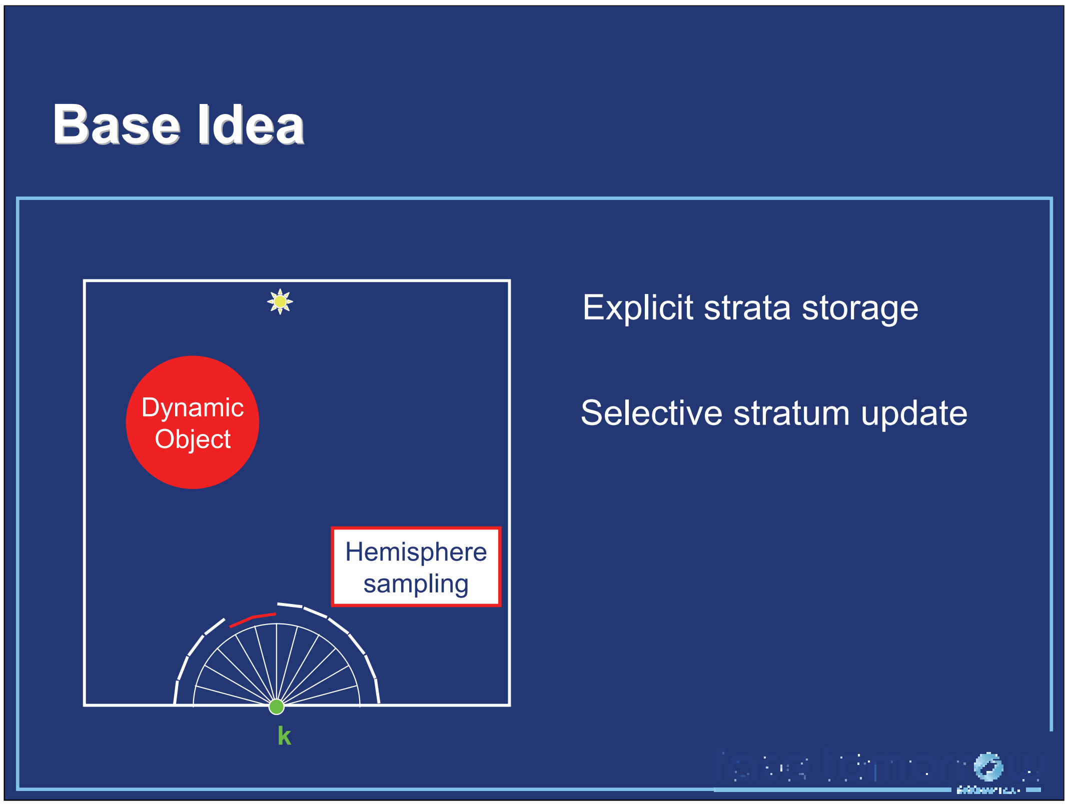


SIGGRAPH2007

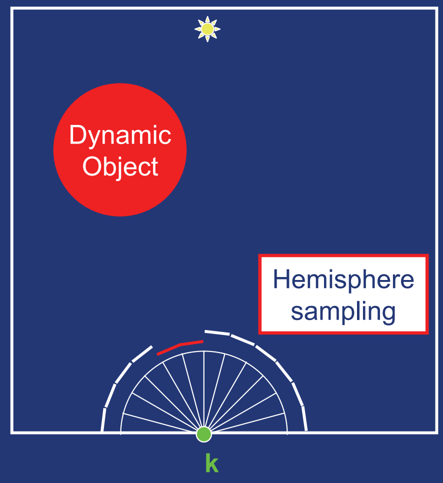
Temporally Coherent Irradiance Caching for High Quality Animation Rendering

Miloslaw Smyk, Shin-ichi Kinuwaki, Roman Durikovic, Karol Myszkowski

In Proceeding of Eurographics 2005



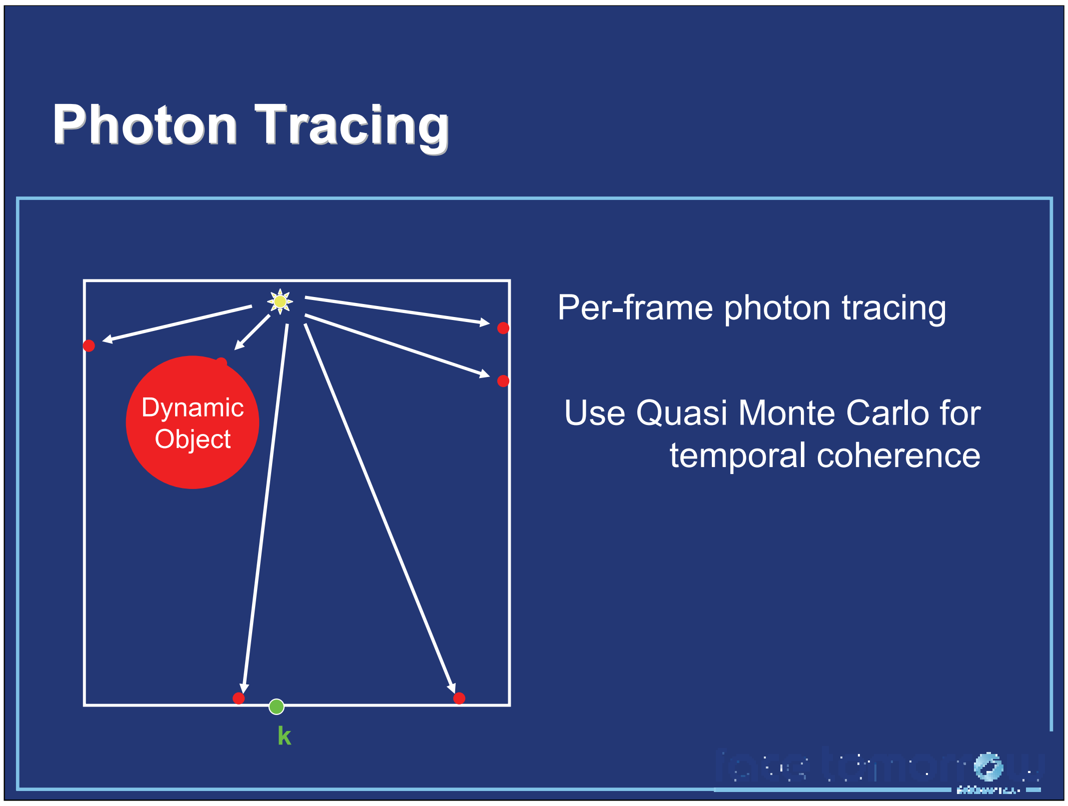
Base Idea



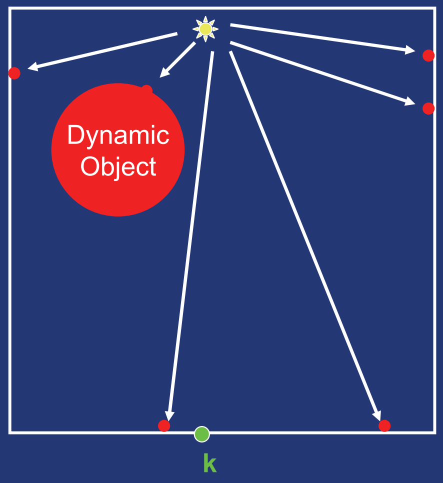
Explicit strata storage

Selective stratum update

The base idea of this paper is the explicit storage and update of all the incoming radiance samples used to estimate the irradiance value of a record. In a dynamic scene, only the stratum corresponding to dynamic objects have to be updated in the course of time, hence reducing the computational cost compared to the classical approach, in which all records are entirely recomputed for each frame of an animation.



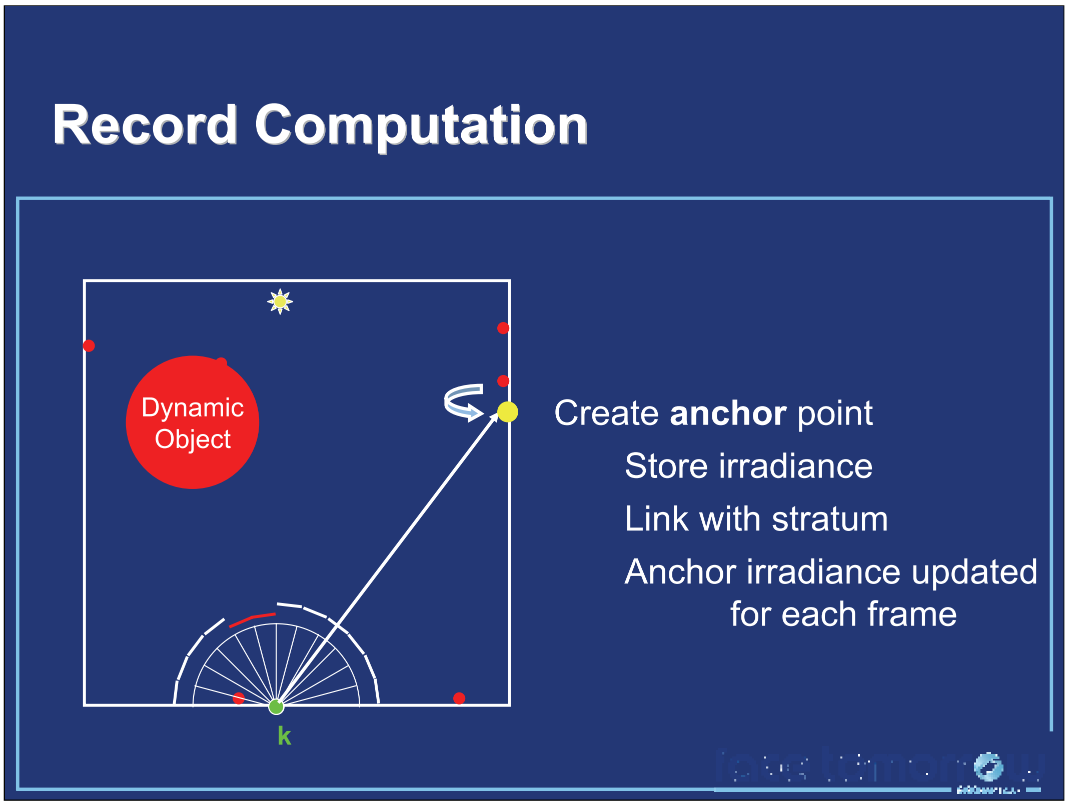
Photon Tracing



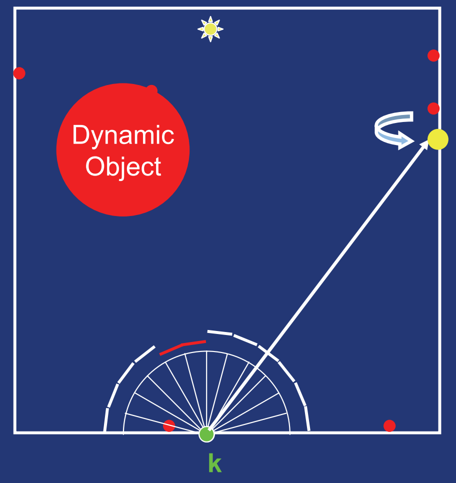
Per-frame photon tracing

Use Quasi Monte Carlo for temporal coherence

For each frame, a photon map is computed by tracing random photons using Quasi Monte Carlo method. Using this method, the paths of the photons are likely to be similar across frames, hence increasing the temporal coherence of the photon map.



Record Computation



Create **anchor** point

Store irradiance

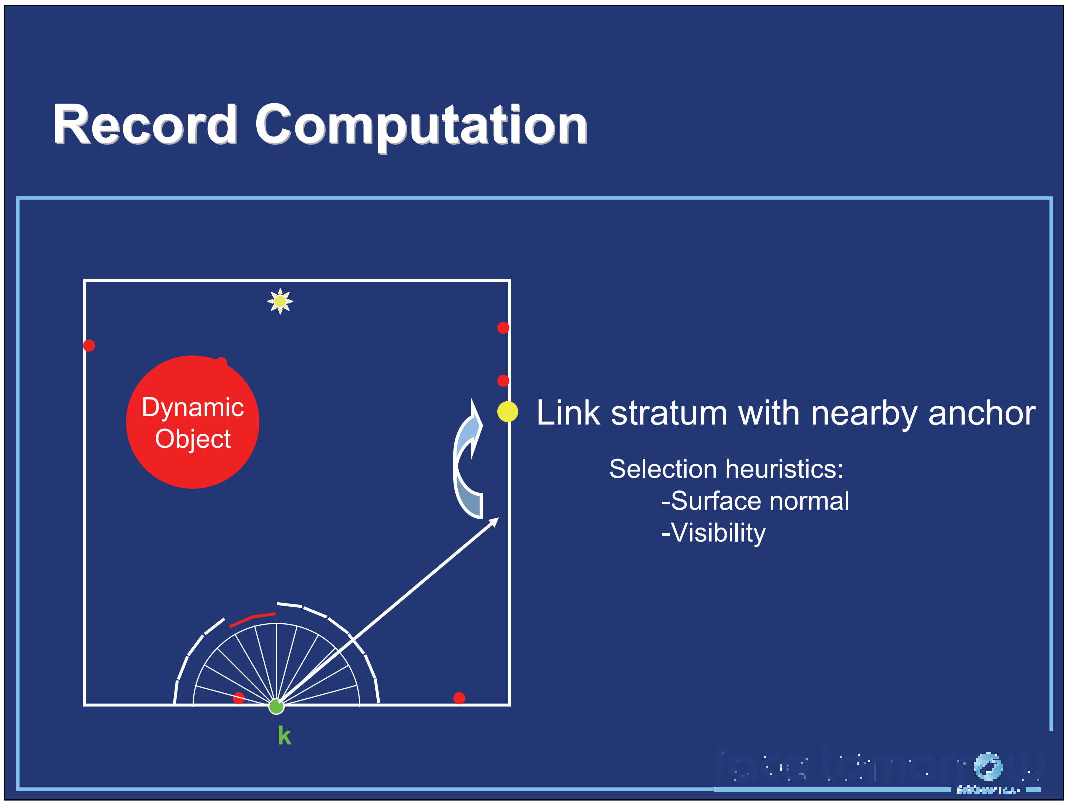
Link with stratum

Anchor irradiance updated for each frame

When creating an irradiance record, a ray is traced for each stratum of the hemisphere. At the intersection point of this ray, an **anchor** point is created. The anchor structure stores the location, the normal, and the irradiance at the intersection point. This anchor irradiance is simply estimated by density estimation on the photon map.

Then, the stratum of the record is explicitly linked to the anchor point.

Note that the location of anchor points is persistent over time. However, their irradiance value is recomputed at each frame using the photon map.

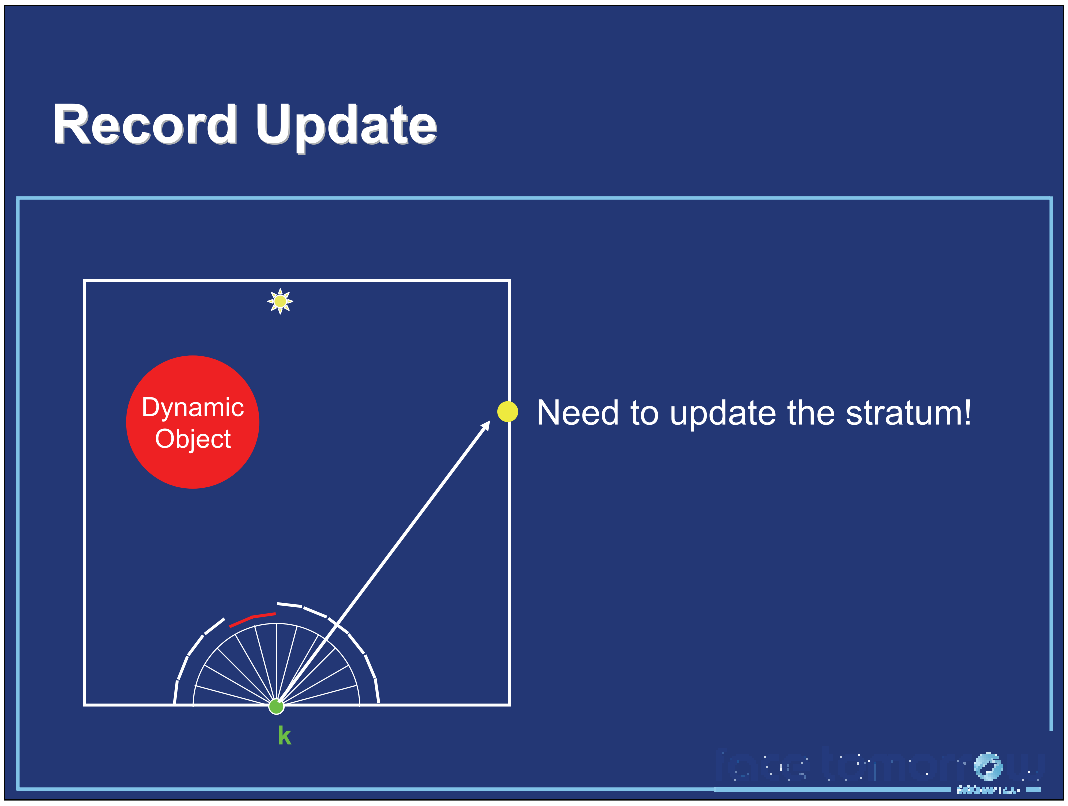


Record Computation

Link stratum with nearby anchor

Selection heuristics:
- -Surface normal
- -Visibility

If the intersection point corresponding to the stratum is close to an existing anchor points, the stratum can be directly linked to the anchor point. Several heuristics have been proposed to estimate the suitability of the link to an existing anchor. Among them, the surface at the intersection point and at the anchor point must have similar normals. Also, the visibility between the record location and the anchor point is explicitly tested by tracing a shadow ray.



Record Update

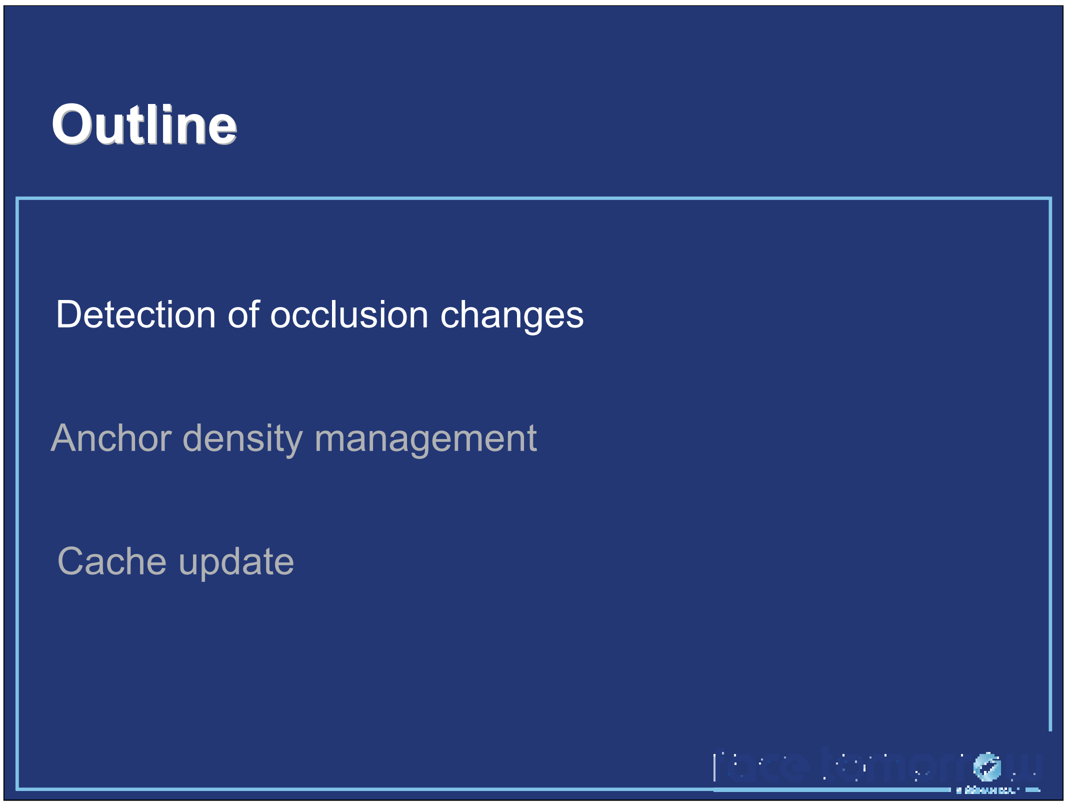
The most important problem to solve in this method is the detection of occlusion changes: when the red sphere crosses the ray between the record and the anchor point, the radiance value of the stratum must be updated accordingly.

Outline

Detection of occlusion changes

Anchor density management

Cache update

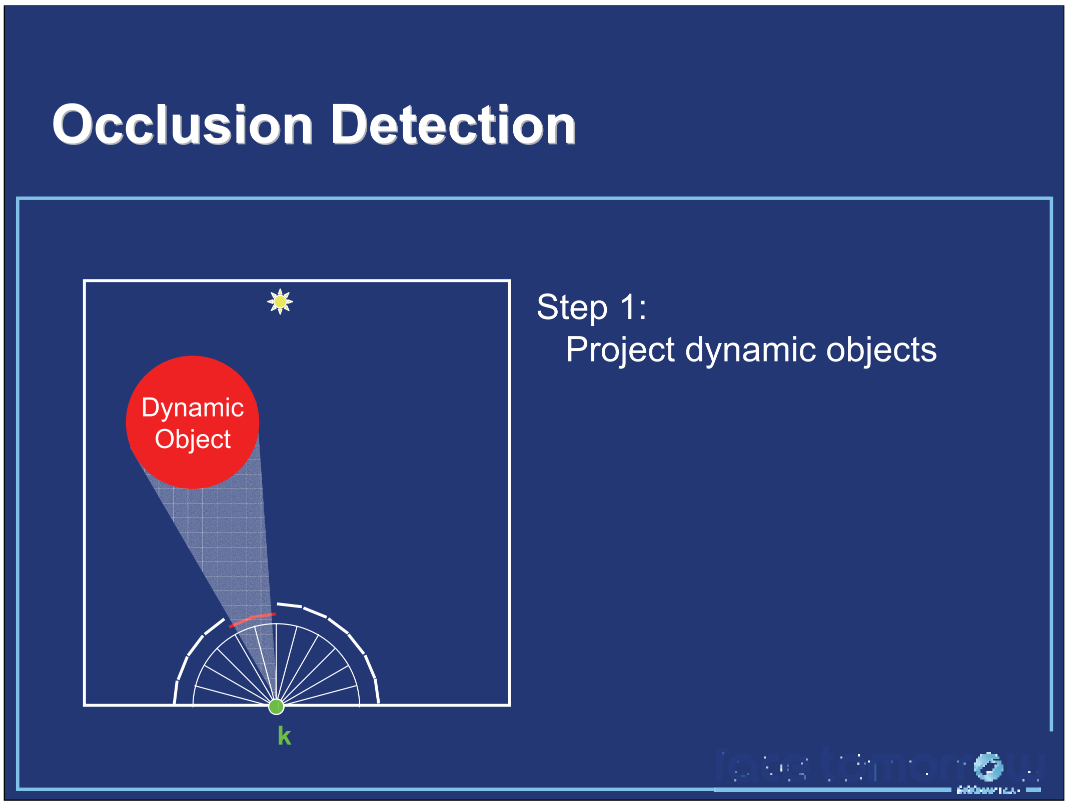


Outline

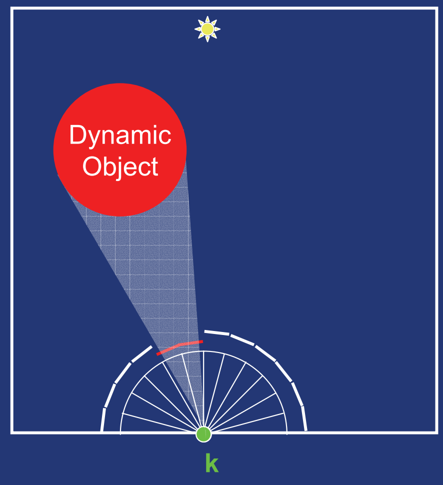
Detection of occlusion changes

Anchor density management

Cache update



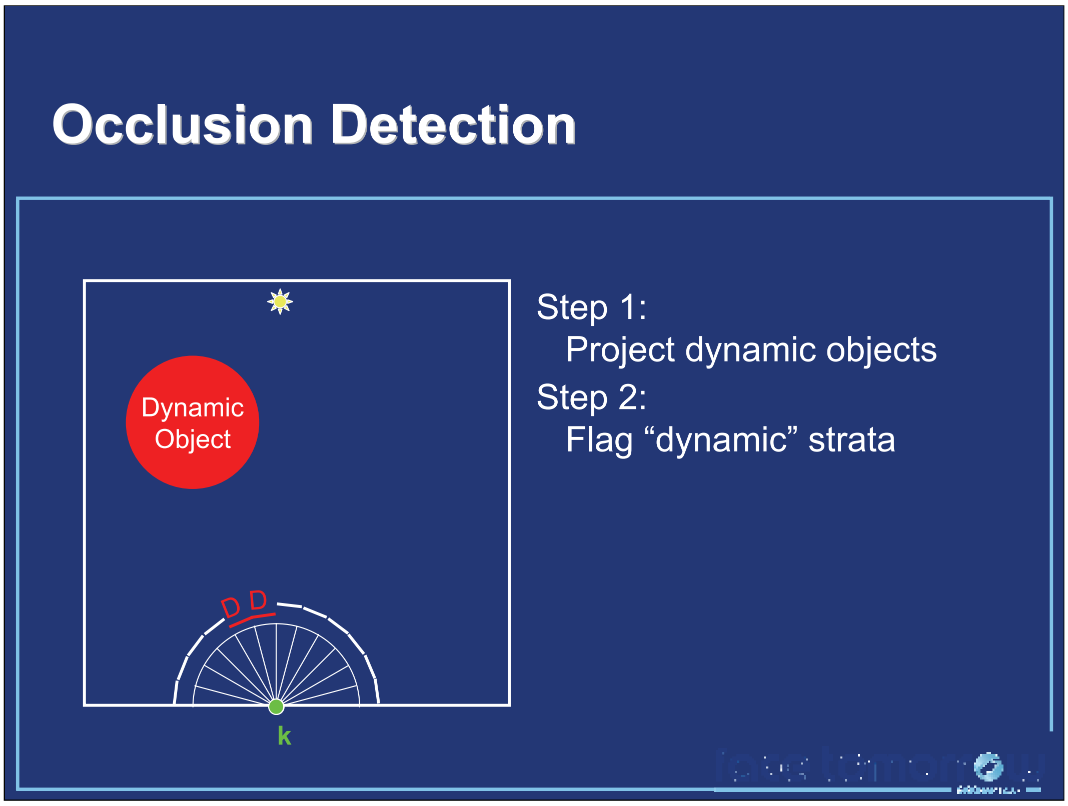
Occlusion Detection



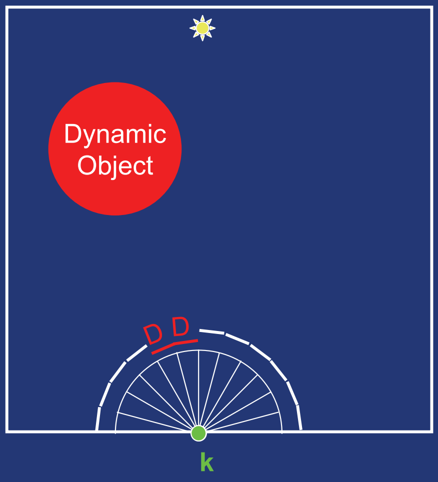
Step 1:
Project dynamic objects

The occlusion detection is performed in 4 steps, and may be performed using graphics hardware.

In the first step, the dynamic objects **only** are projected onto the hemisphere above the record.

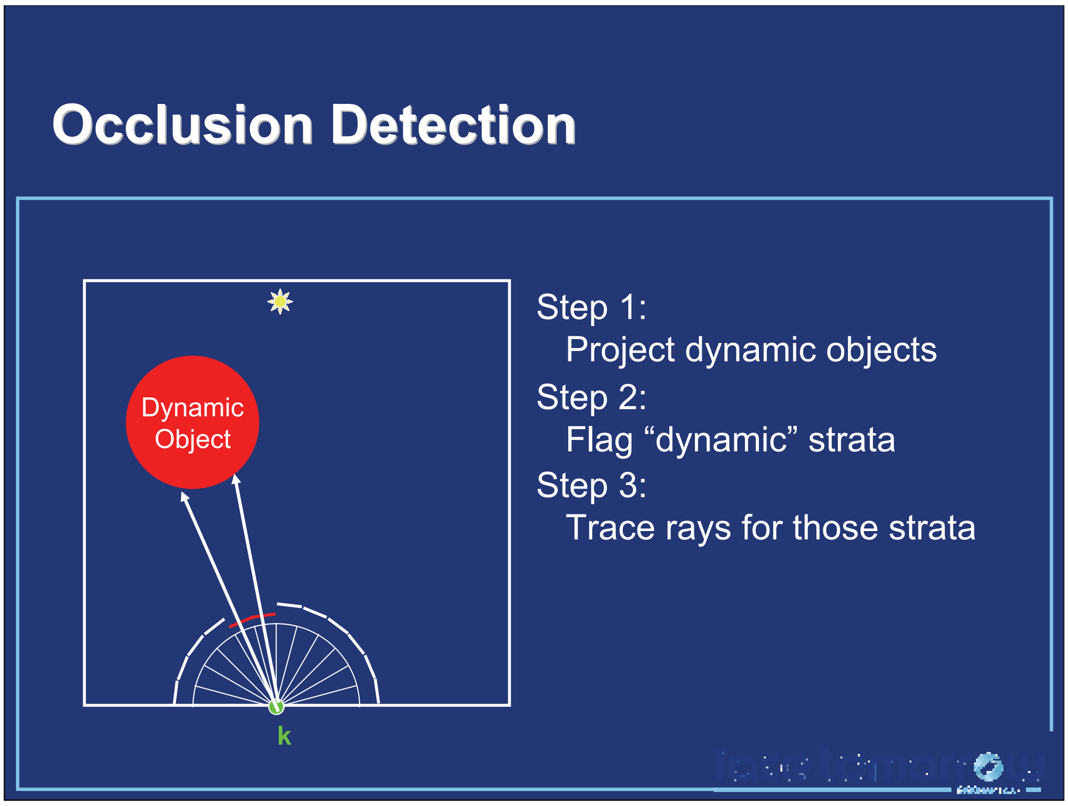


Occlusion Detection

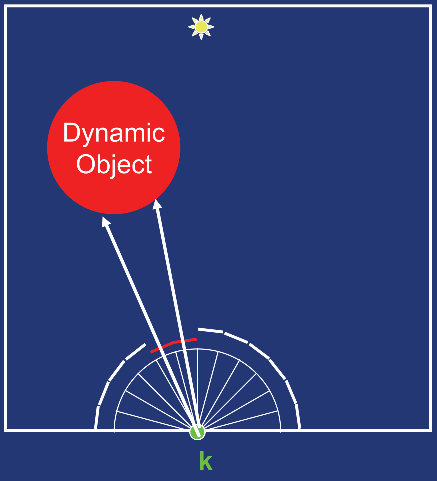


Step 1:
Project dynamic objects
Step 2:
Flag “dynamic” strata

This projection of the dynamic objects allows us to flag the corresponding strata as « dynamic strata », e.g. strata for which the incoming radiance is due to a dynamic object.

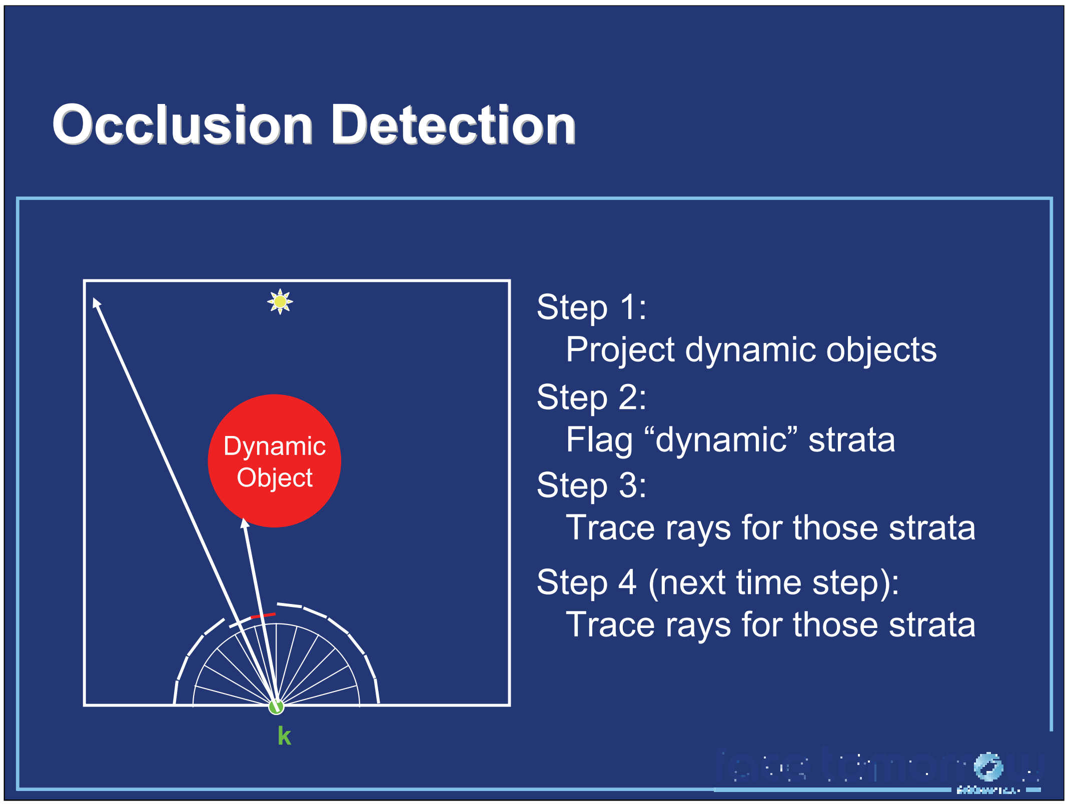


Occlusion Detection

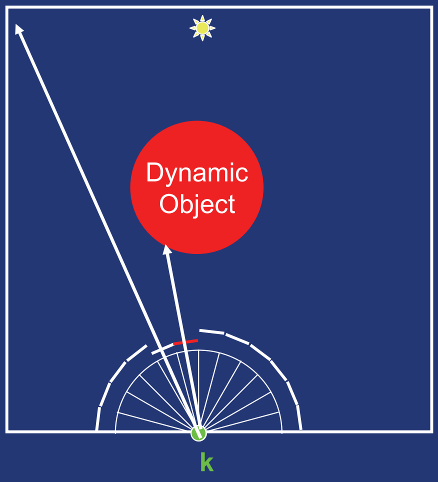


Step 1:
  Project dynamic objects
Step 2:
  Flag “dynamic” strata
Step 3:
  Trace rays for those strata

For those strata, rays are traced to estimate the incoming radiance. Note that anchor points are not created on dynamic objects. Instead, the irradiance at each intersection point is computed by density estimation in the photon map. Combined with the existing information contained in « static » strata, the irradiance of the record can be easily deduced.



Occlusion Detection



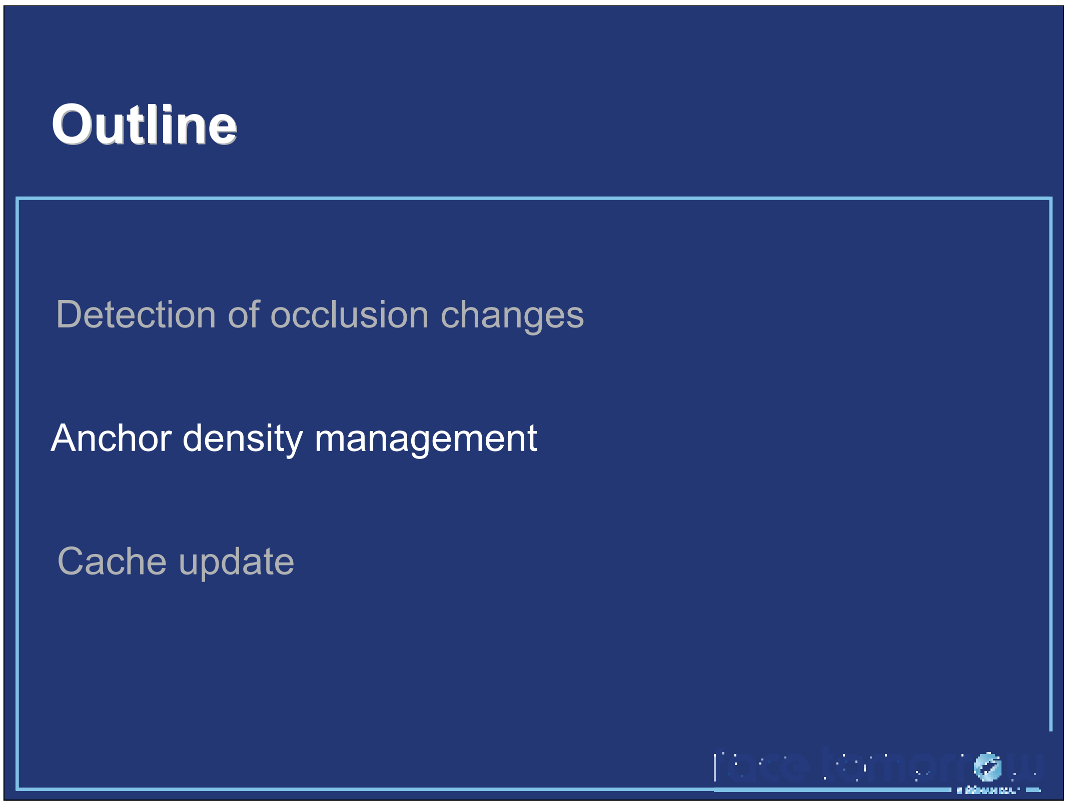
Step 1:
  Project dynamic objects

Step 2:
  Flag “dynamic” strata

Step 3:
  Trace rays for those strata

Step 4 (next time step):
  Trace rays for those strata

At the next time step, the « dynamic strata » will also be explicitly sampled by ray tracing to update their values.



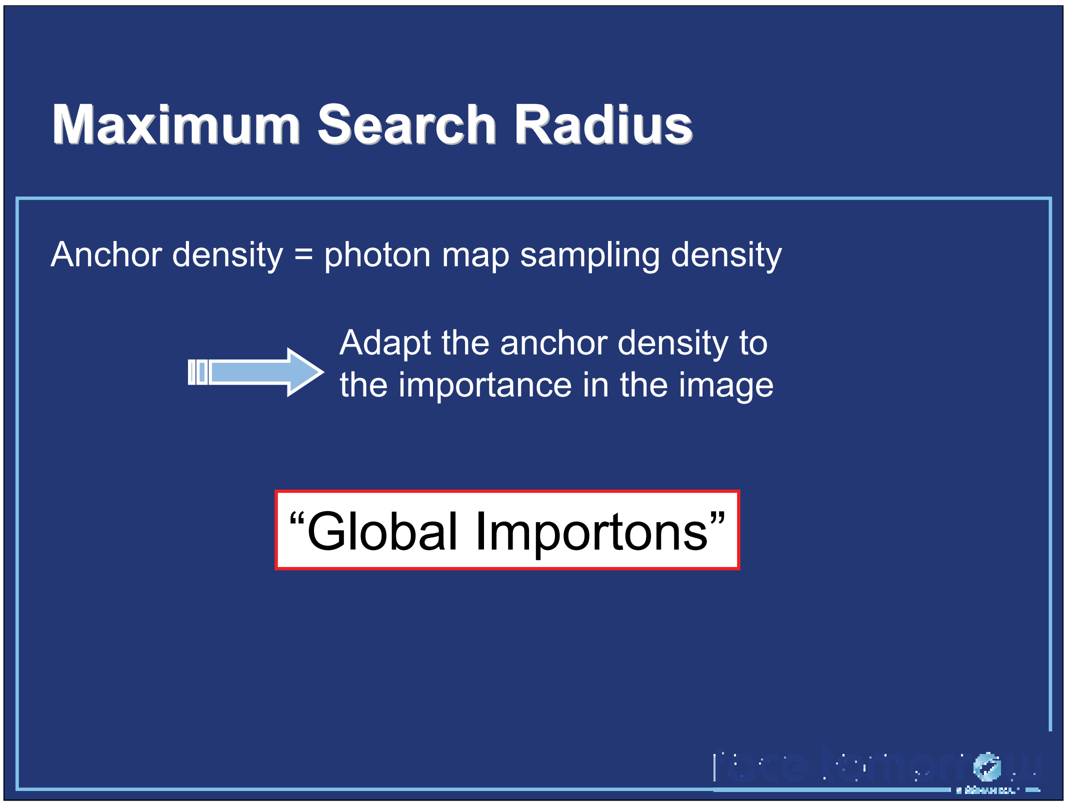
Outline

Detection of occlusion changes

Anchor density management

Cache update

The anchor data structure allows us to reduce the number of density estimations to render an animation. However, the efficiency of the method lies in an adaptive distribution of anchors.



Maximum Search Radius

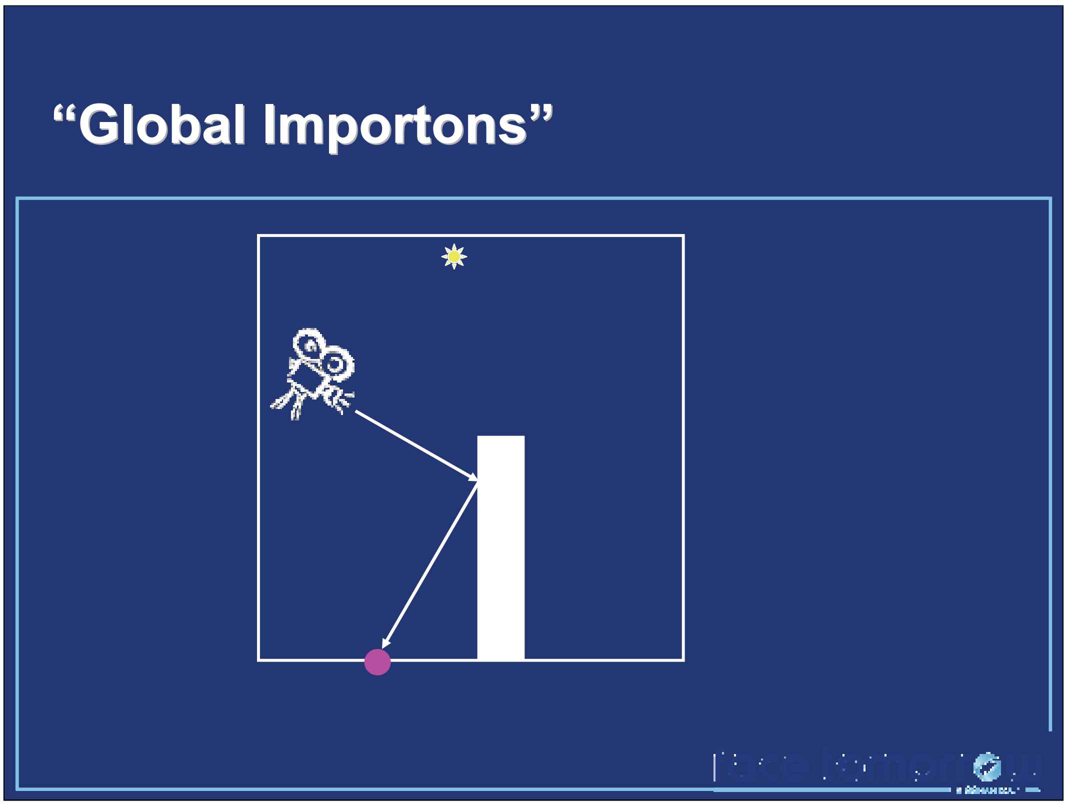
Anchor density = photon map sampling density

➡ Adapt the anchor density to the importance in the image

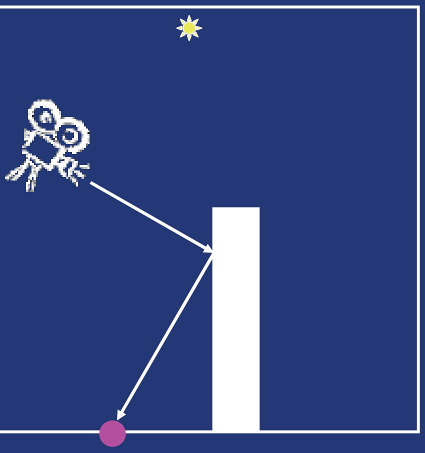
“Global Importons”

The density of anchor points directly impacts the quality of the lighting estimate. In particular, the distribution of anchor points should be very dense at points which highly contribute to the lighting of visible points. This density is controlled by the maximum search radius used when looking for an anchor nearby an intersection point. If this radius is small, the queries in the anchor structure may not find any suitable nearby anchor. Therefore, a small radius increases the anchor density. Converserly, a large radius decreases the anchor density.

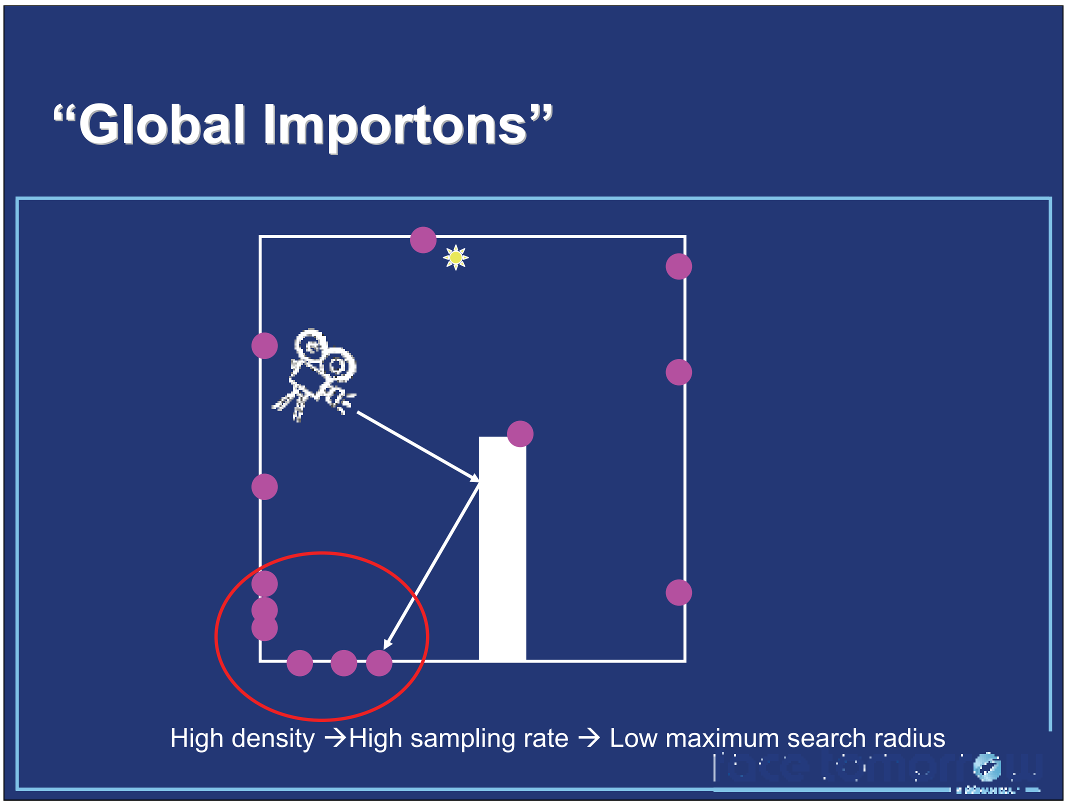
The authors use “global importons” to adapt the density of anchors.



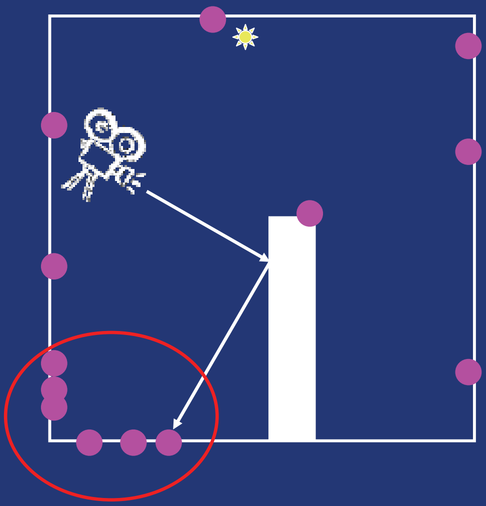
“Global Importons”



The “global importons” are obtained by tracing rays from the camera, and storing the second bounce of the ray in the scene.

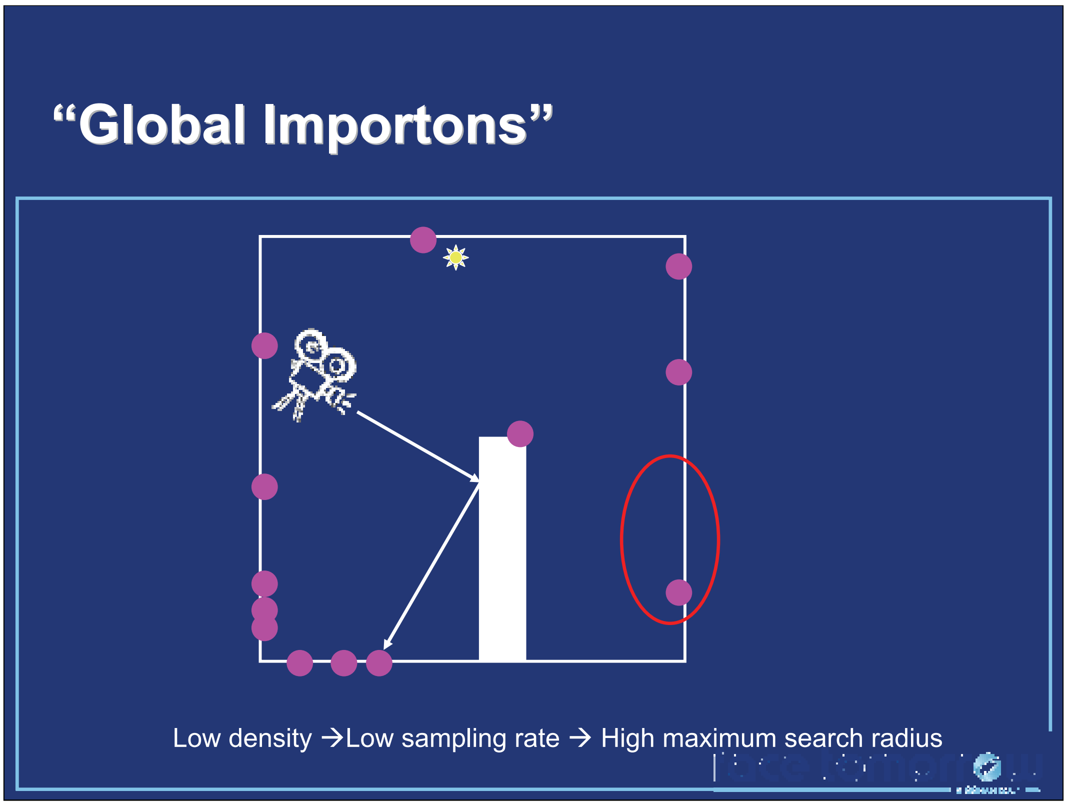


“Global Importons”



High density → High sampling rate → Low maximum search radius

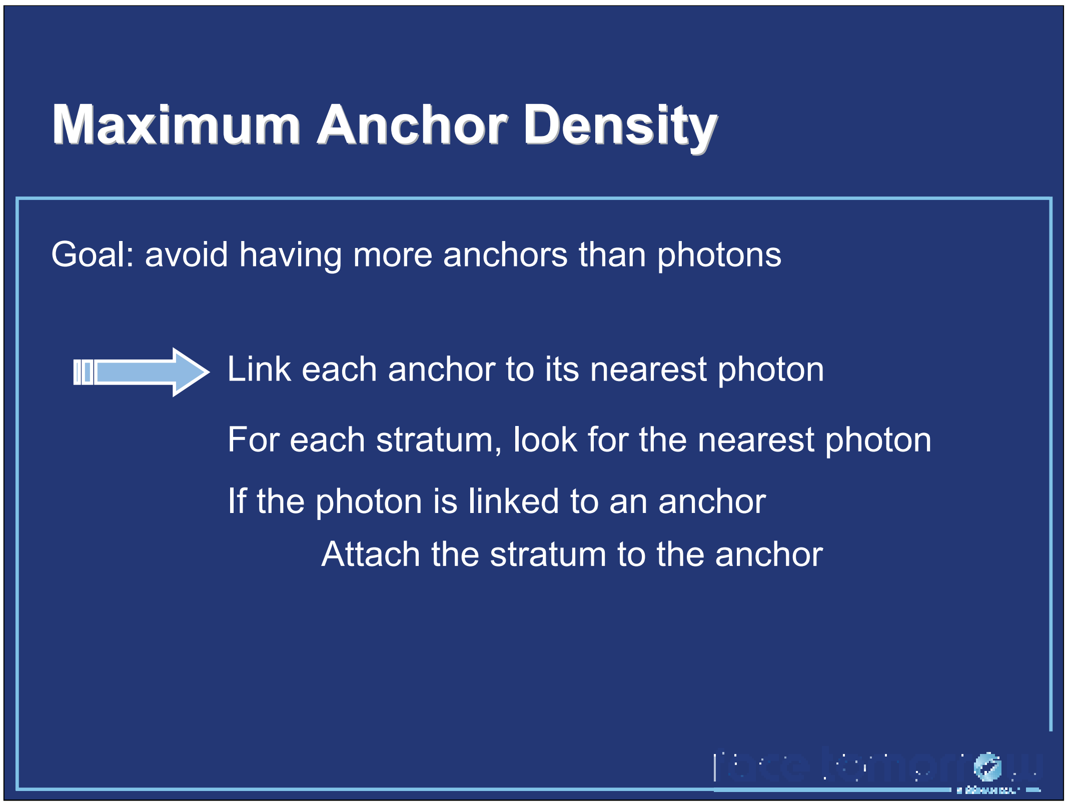
The zones containing many “global importons” are zones which highly contribute to the indirect lighting of visible points. Therefore, the anchor density should be raised to ensure the image quality. Therefore, the maximum search radius is lowered.



“Global Importons”

Low density →Low sampling rate → High maximum search radius

Conversely, zones with very few importons do not contribute much to the indirect lighting of visible points. Therefore, the sampling rate is kept low to increase the rendering speed. The maximum search distance is then set to a high value.



Maximum Anchor Density

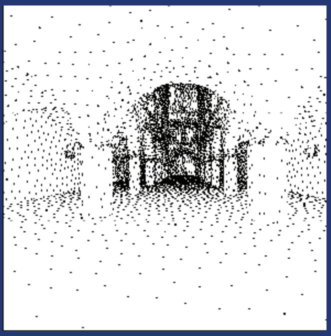
Goal: avoid having more anchors than photons

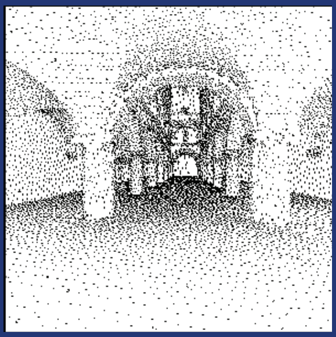
➡ Link each anchor to its nearest photon

```
For each stratum, look for the nearest photon
If the photon is linked to an anchor
        Attach the stratum to the anchor
```

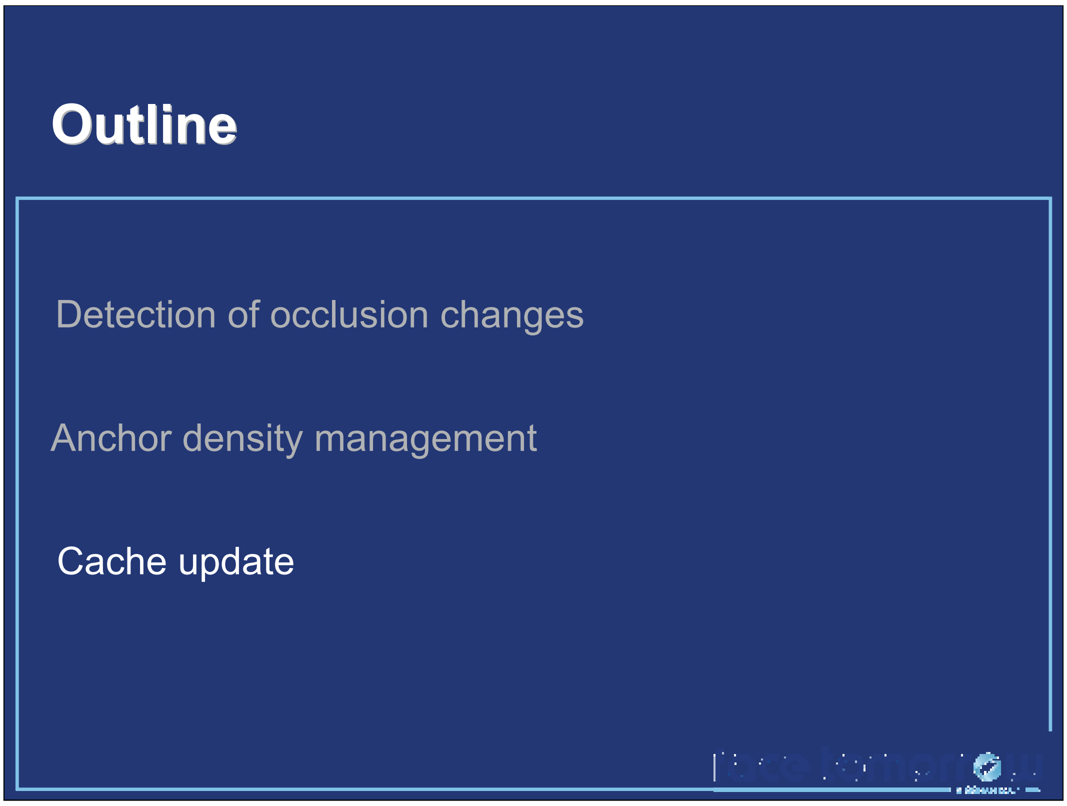
One of the goals of this work is the reduction of the photon searches during global illumination computation. Therefore, if the structure contains more anchors than photons, the algorithm would become very inefficient. The method described in this slide ensures that there cannot be more anchors than photons

Anchor Density Management


No density management


With density management

(Images courtesy of Miloslaw Smyk et al.)

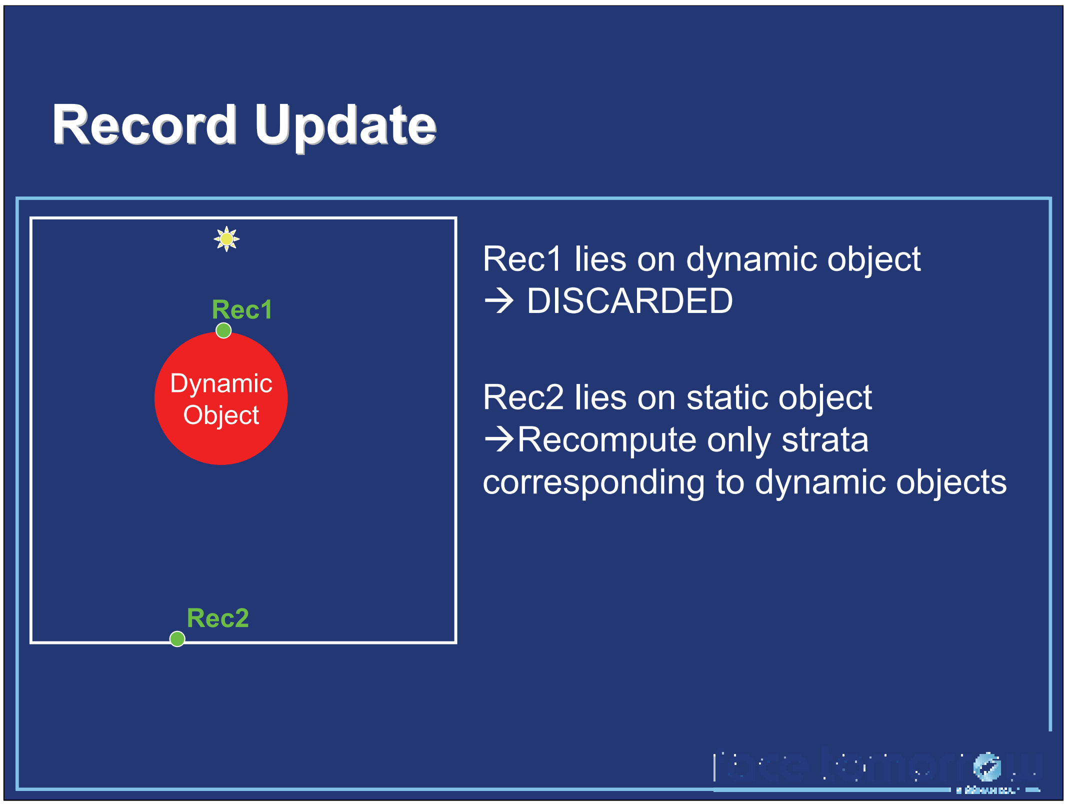


Outline

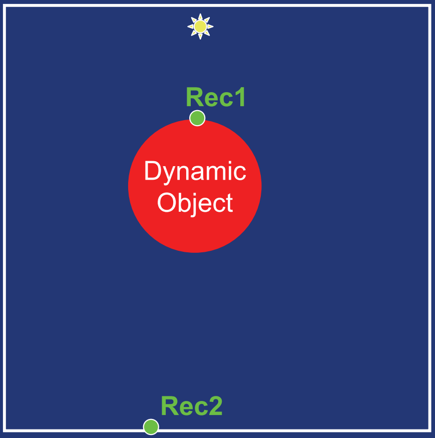
Detection of occlusion changes

Anchor density management

Cache update



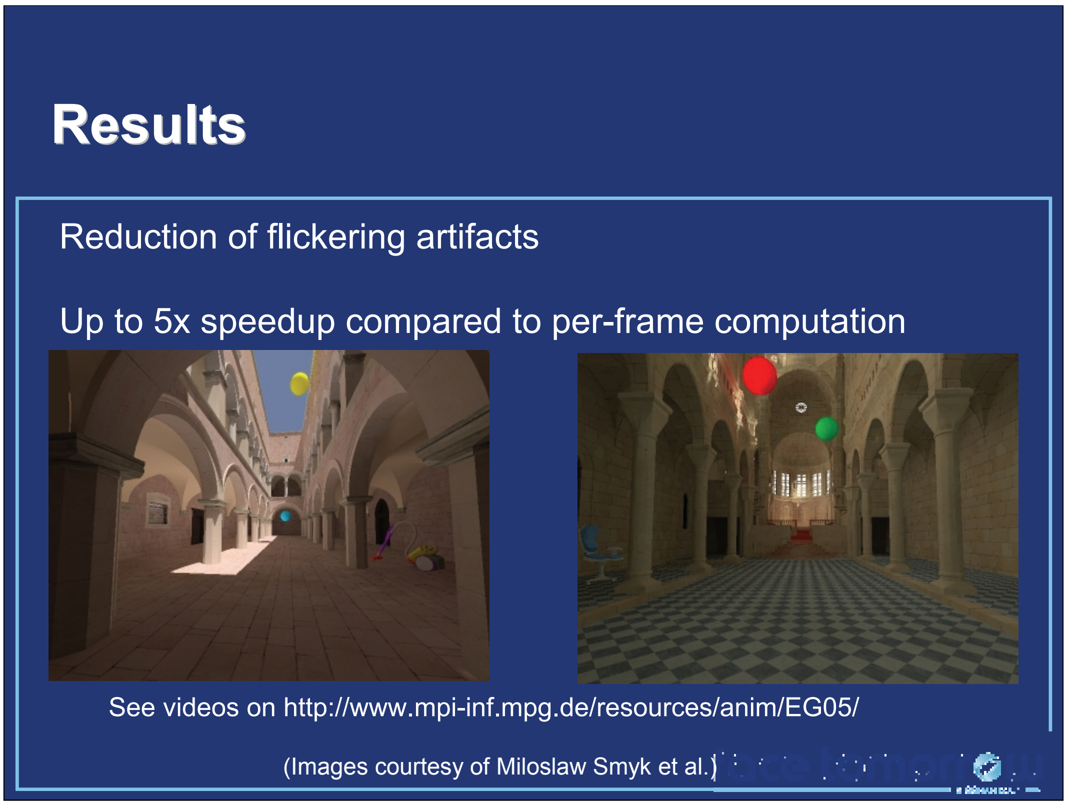
Record Update



Rec1 lies on dynamic object
→ DISCARDED

Rec2 lies on static object
→Recompute only strata corresponding to dynamic objects

The irradiance value of the records must be updated across frames. Several cases can happen: records located on dynamic objects are discarded for each frame. Records on static objects undergo adaptive strata replacement as explained previously.



Results

Reduction of flickering artifacts

Up to 5x speedup compared to per-frame computation

See videos on http://www.mpi-inf.mpg.de/resources/anim/EG05/

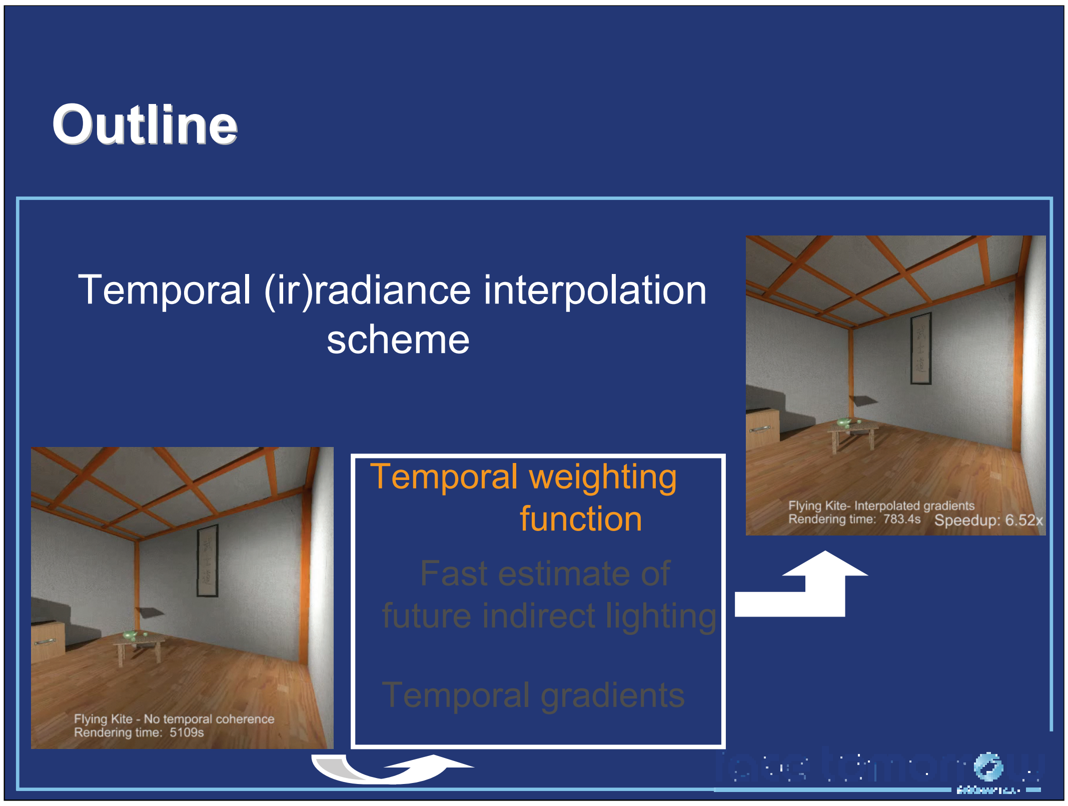
(Images courtesy of Miloslaw Smyk et al.)

The resulting animations exhibit a significant reduction of flickering, while reducing the computational cost by a factor up to 5.

Temporal Radiance Caching

Pascal Gautron, Kadi Bouatouch, Sumanta Pattanaik

To appear in IEEE Transactions on Visualization and Computer Graphics



Outline

Temporal (ir)radiance interpolation scheme

Temporal weighting function

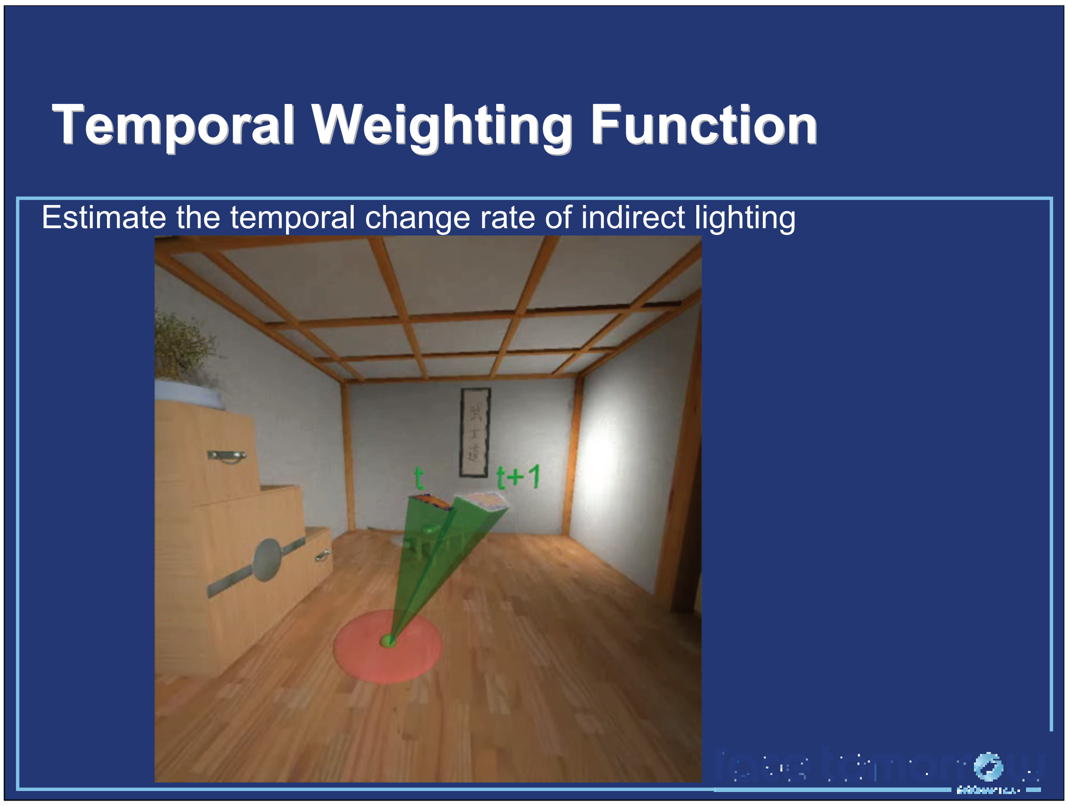
Fast estimate of future indirect lighting

Temporal gradients

Flying Kite - No temporal coherence
Rendering time: 5109s

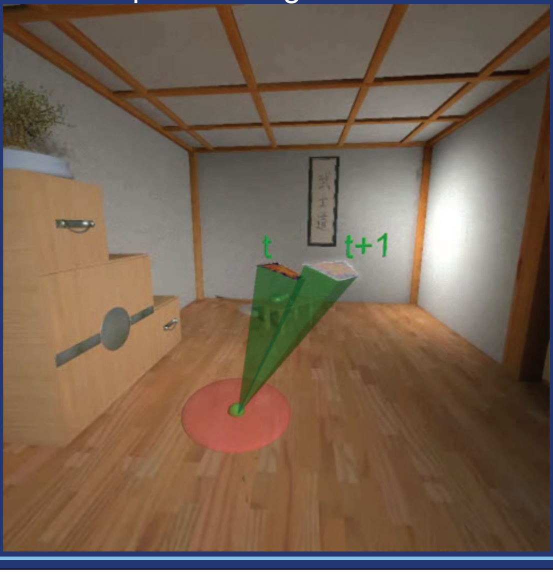
Flying Kite- Interpolated gradients
Rendering time: 783.4s Speedup: 6.52x

More precisely, the method described in this course is an extension of the irradiance caching interpolation scheme to the temporal domain. To this end, we devise a temporal weighting function to determine the lifespan of a record, e.g. the number of frames in which a record can be reused without degrading the rendering quality. The contribution of a record within its lifespan is estimated using temporal irradiance gradients.

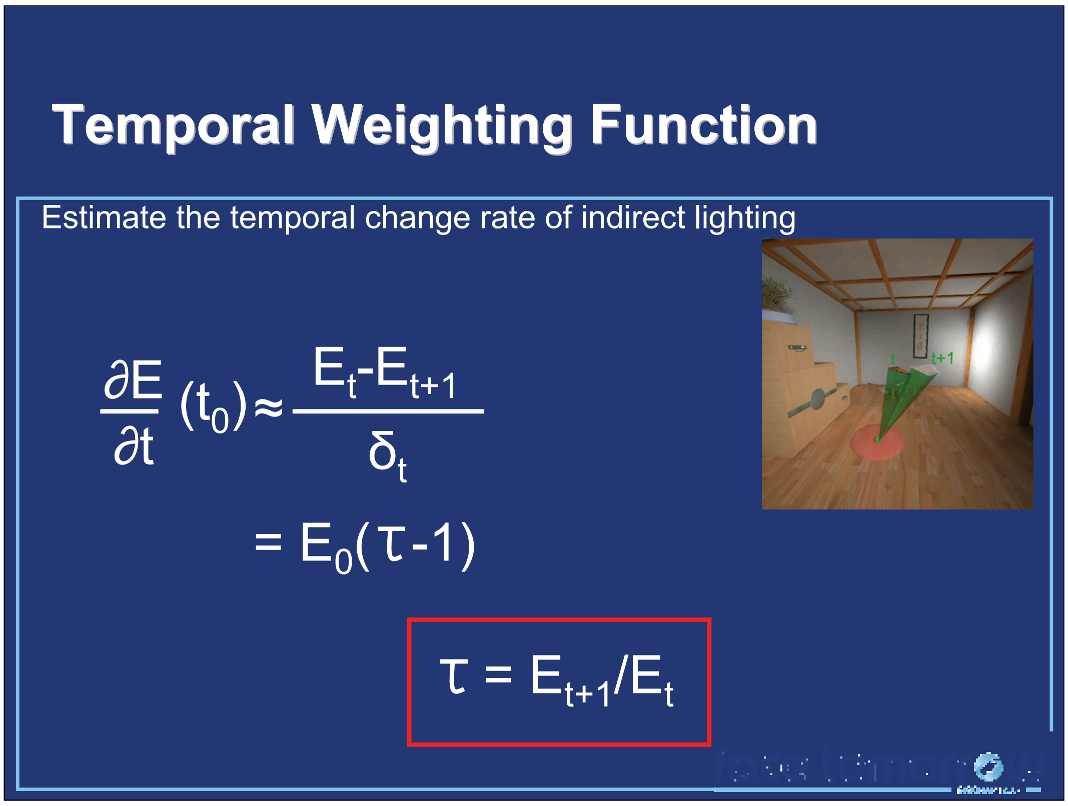


Temporal Weighting Function

Estimate the temporal change rate of indirect lighting



The spatial weighting function described in [Ward88] is based on an estimate of the change of indirect lighting with respect to displacement and rotation. The temporal weighting function is thus based on an estimate of the **temporal change** of indirect lighting across frames.



Temporal Weighting Function

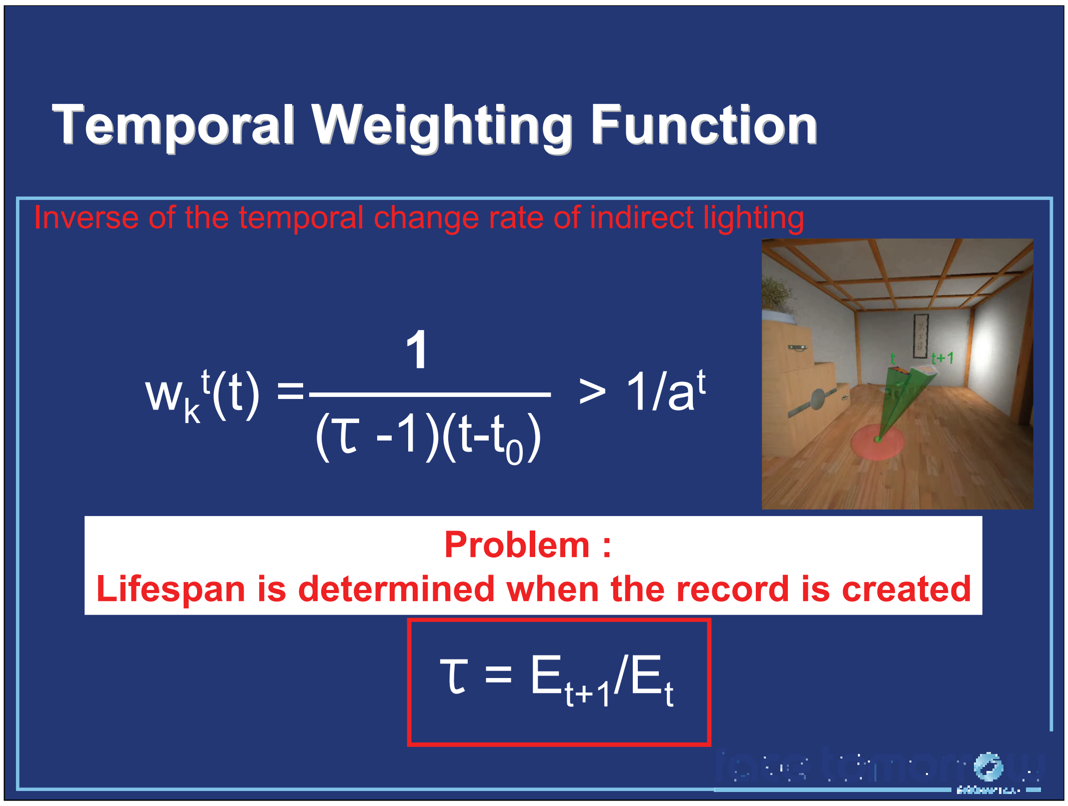
Estimate the temporal change rate of indirect lighting

$$\frac{\partial E}{\partial t}(t_0) \approx \frac{E_t - E_{t+1}}{\delta_t}$$

$$= E_0(\tau - 1)$$

$$\boxed{\tau = E_{t+1}/E_t}$$

Let us consider the irradiance E_t at current frame, and the irradiance E_{t+1} at next frame. The temporal change of indirect lighting can be estimated by a numerical estimation of the temporal derivative of the lighting. We define the value τ as the ratio of the future and current lighting.



Temporal Weighting Function

Inverse of the temporal change rate of indirect lighting

$$w_k^t(t) = \frac{1}{(\tau - 1)(t - t_0)} > 1/a^t$$

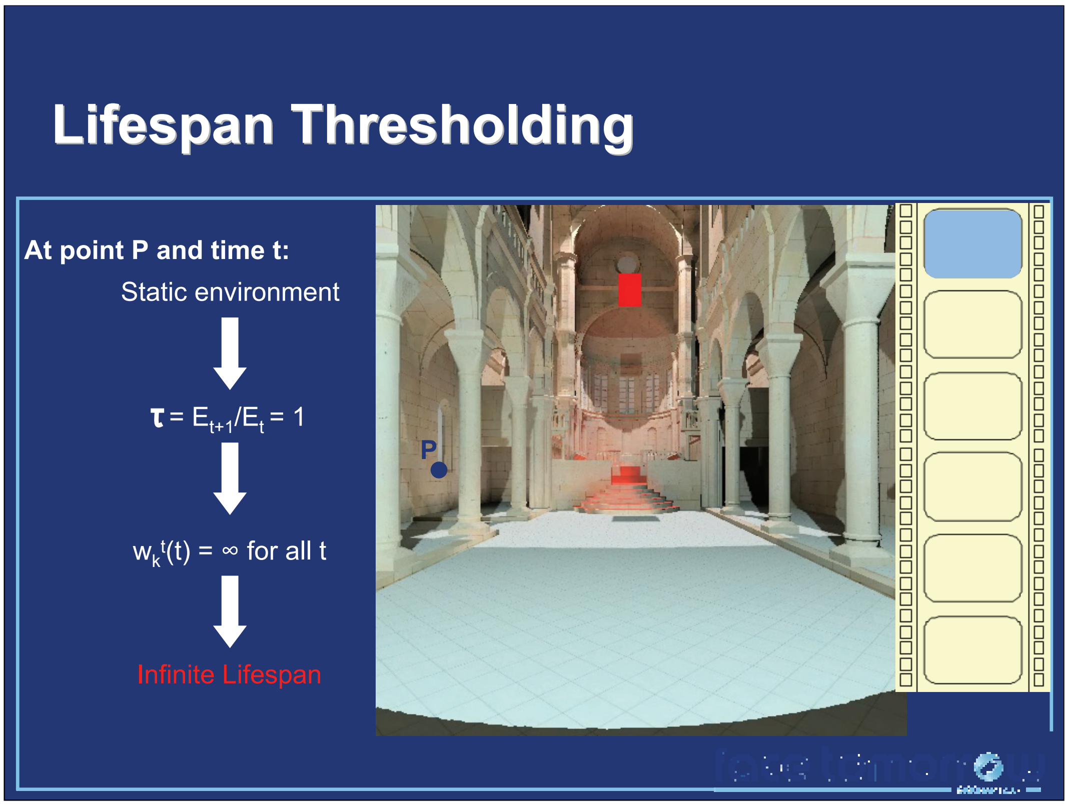
Problem :
Lifespan is determined when the record is created

$$\boxed{\tau = E_{t+1}/E_t}$$

Using a derivation similar to [Ward88], we obtain a temporal weighting function defined as the inverse of the change of indirect lighting over time. The lifespan of a record is thus adapted to the **local change** of indirect lighting: fast changes yield a short lifespan, while records can be reused across many frames if the temporal changes are slow. The user-defined threshold value a^t conditions the temporal accuracy of the computation: a high value leads to long lifespans, hence reducing the rendering time at the detriment of the quality. Conversely, a small value ensures frequent updates at the expense of rendering time.

At the end of its lifespan, a record is discarded and replaced by a new record located at the same point. This method strenghtens the temporal coherence of the distribution of records, hence avoiding flickering artifacts due to changes of record distribution.

However, the lifespan is determined using the indirect lighting at current and next frames only. Therefore, later changes do not affect this lifespan, hence harming the reactivity of the algorithm. An overestimation of the lifespan yields residual global illumination effects known as « ghosting artifacts ».



Lifespan Thresholding

At point P and time t:

Static environment

↓

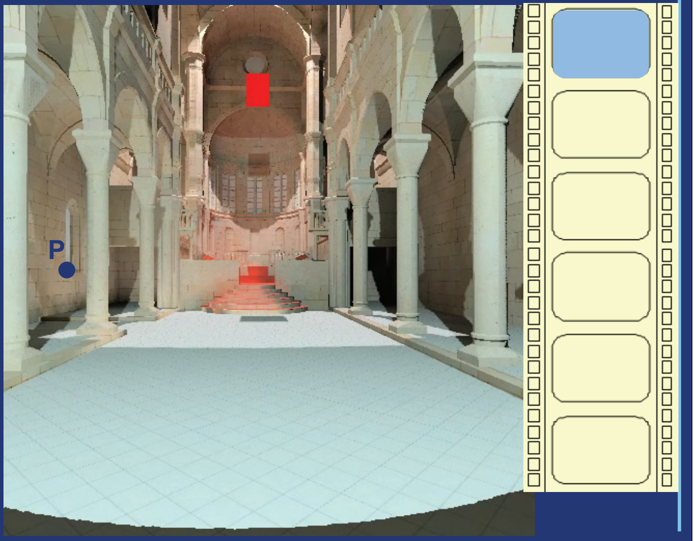
$\tau = E_{t+1}/E_t = 1$

↓

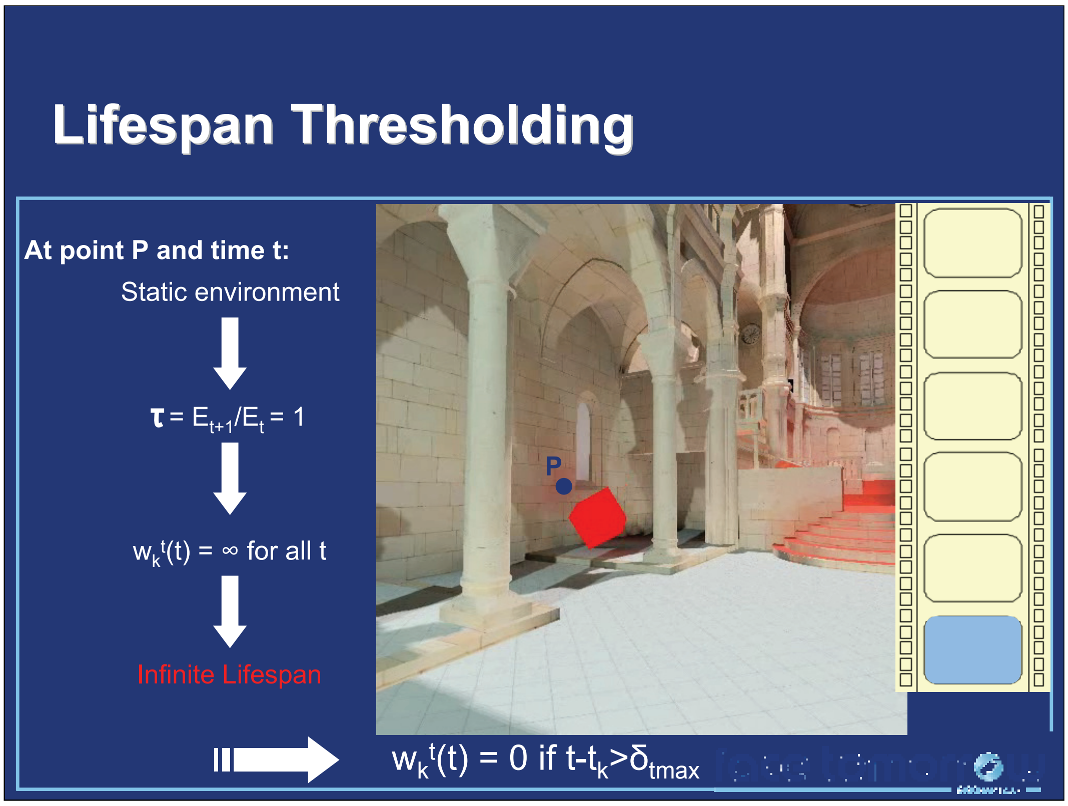
$w_k^t(t) = \infty$ for all t

↓

Infinite Lifespan



Let us consider an example: at a point P and time t and t+1, the environment is static. Therefore, the indirect lighting is considered constant. In this case, the value of the temporal weighting function is infinite for any frame. The lifespan of the record is then considered infinite.



Lifespan Thresholding

At point P and time t:

Static environment

↓

$\tau = E_{t+1}/E_t = 1$

↓

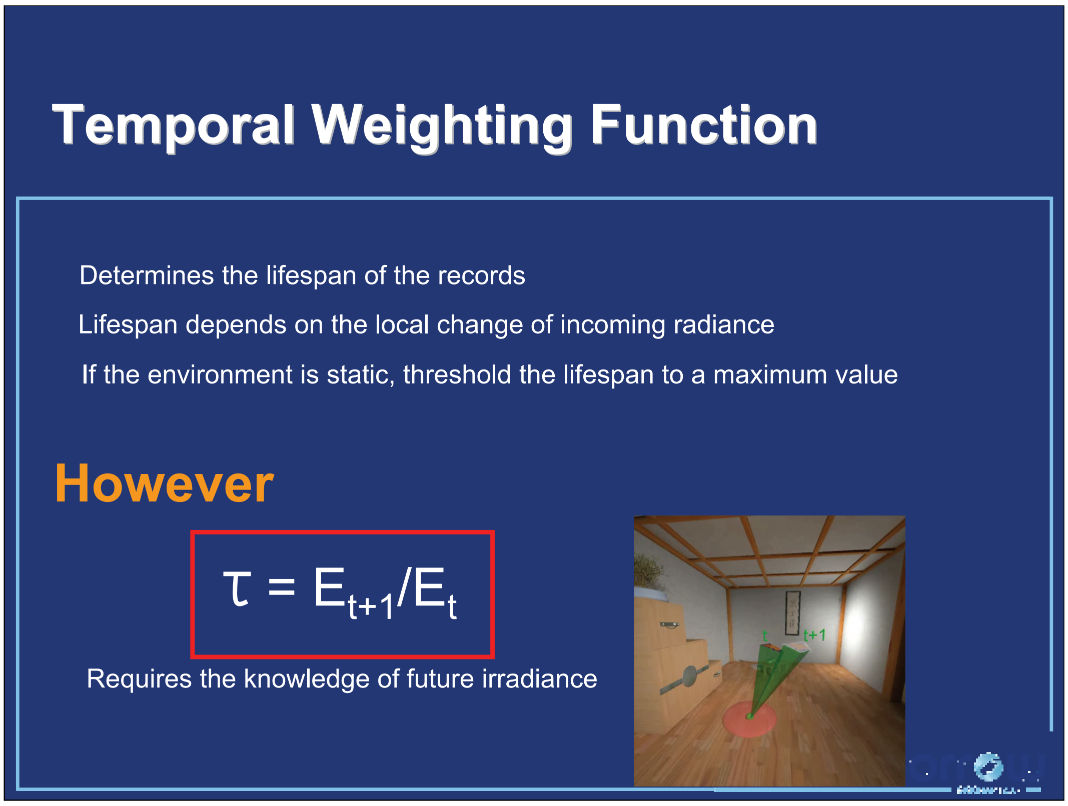
$w_k^t(t) = \infty$ for all t

↓

Infinite Lifespan

⇒ $w_k^t(t) = 0$ if $t - t_k > \delta_{tmax}$

However, it is easy to show that the lighting at P may change afterwards. Since the weighting function returns an infinite value regardless of the frame, the value of the lighting at P will never be updated, yielding ghosting artifacts. We do not solve this problem, but we simply ask the user to define a maximum lifespan for all records. This maximum lifespan ensures the update of the ligthing at least after



Temporal Weighting Function

Determines the lifespan of the records

Lifespan depends on the local change of incoming radiance

If the environment is static, threshold the lifespan to a maximum value

However

$$\boxed{\tau = E_{t+1}/E_t}$$

Requires the knowledge of future irradiance

The temporal weighting function is used to determine the lifespan of each record based on the local change of incoming radiance over time. The length of the lifespan is shortened or lengthened with respect to the magnitude of the change of indirect lighting. However, the estimate of this change is based on the knowledge of the indirect lighting at next frame, E_{t+1}. Since this value is generally unknown, next section will devise a method for fast estimation of the future incoming without actual sampling.

Contributions

Temporal (ir)radiance interpolation scheme

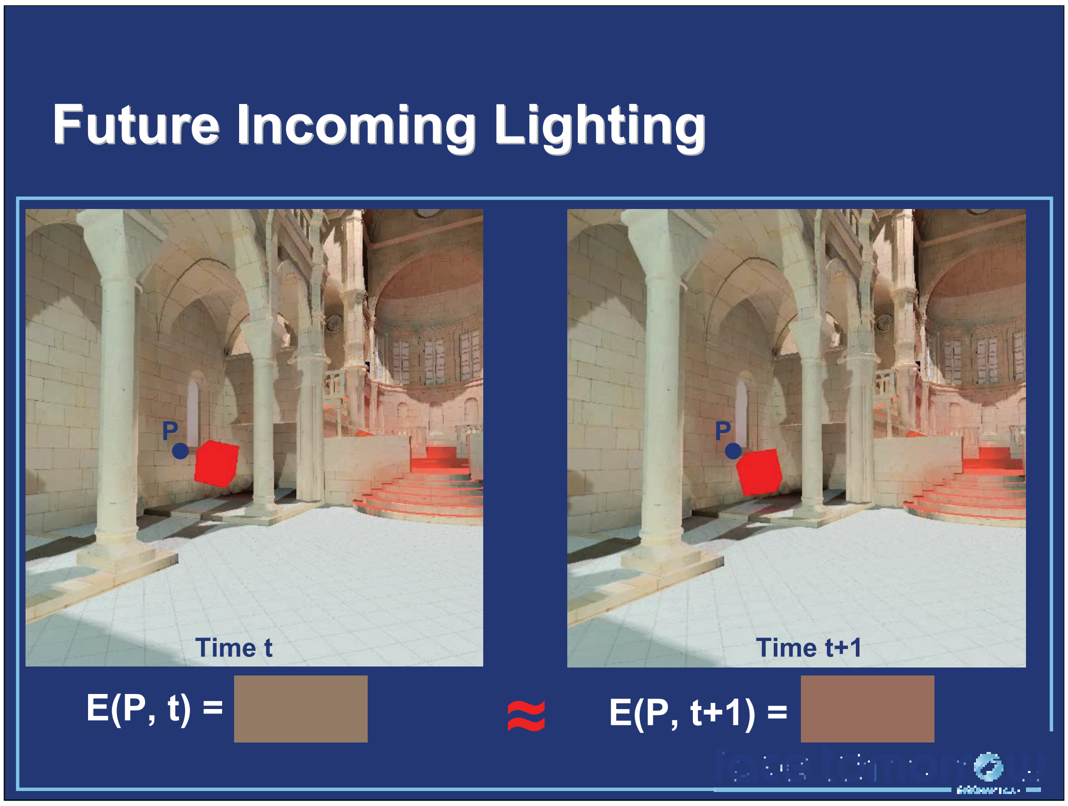
Flying Kite- Interpolated gradients
Rendering time: 783.4s
Speedup: 6.52x

Flying Kite - No temporal coherence
Rendering time: 5109s

Temporal weighting function

Fast estimate of future indirect lighting

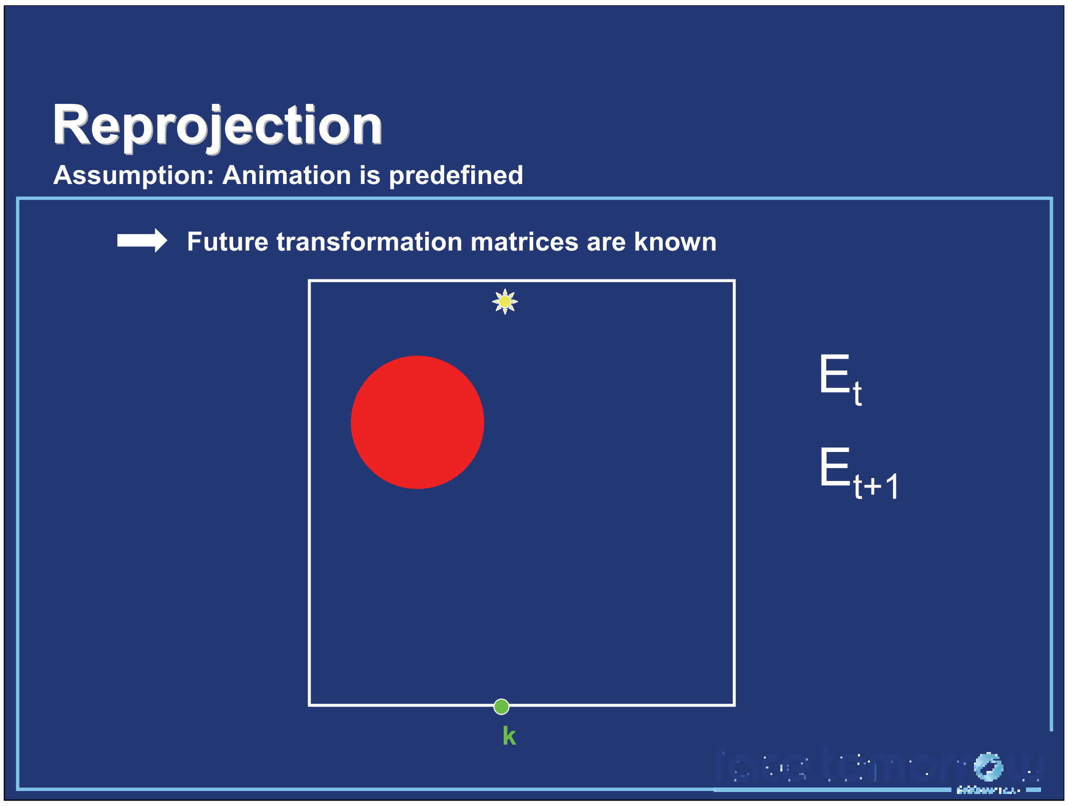
Temporal gradients



Future Incoming Lighting

$E(P, t) =$ ▨ $\approx$ $E(P, t+1) =$ ▨

This approach is based on the following observation: between time t and t+1, the change of lighting is low. Based on the knowledge of the dynamic properties of surrounding objects, we propose a method based on reprojection to estimate the future incoming lighting using only the data available at current frame.



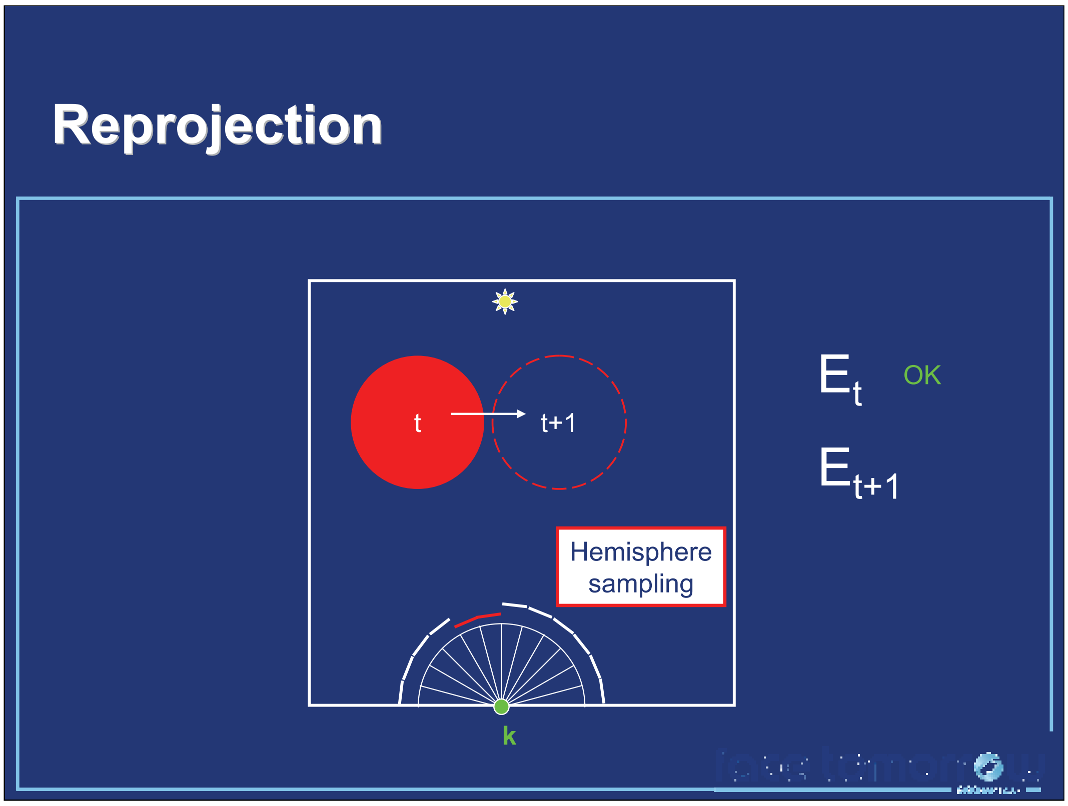
Reprojection

Assumption: Animation is predefined

➡ Future transformation matrices are known

Our reprojection method is similar to the one used in the Render Cache [Walter99]. However, in the Render Cache, the reprojection is used to avoid tracing primary rays from the camera. In our case, the reprojection is only used to estimate the future incoming radiance at the location of a record.

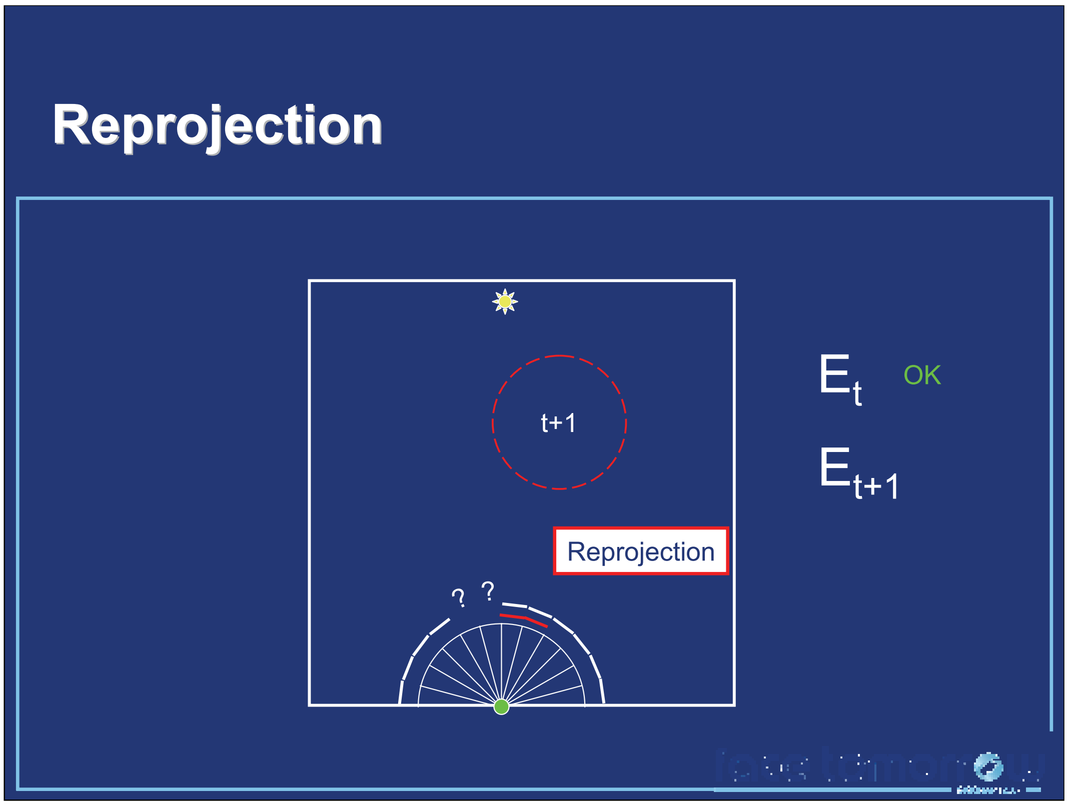
Let us consider a point k at which we want to create a record.



Reprojection

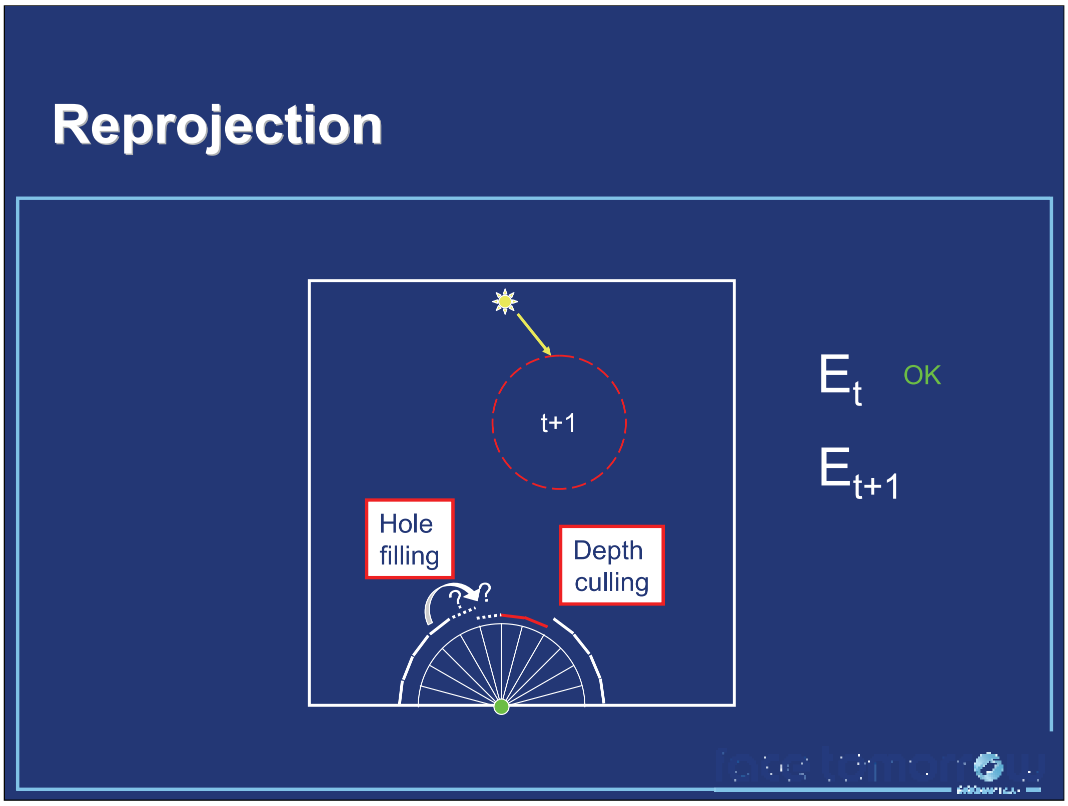
The first step is the computation of the incoming radiance in k at time t. This is performed by sampling the hemisphere above k either using ray tracing or GPU rasterization as explained in the previous chapter.

Assuming the animation is known in advance, the position of the red sphere at time t+1 is known.



Reprojection

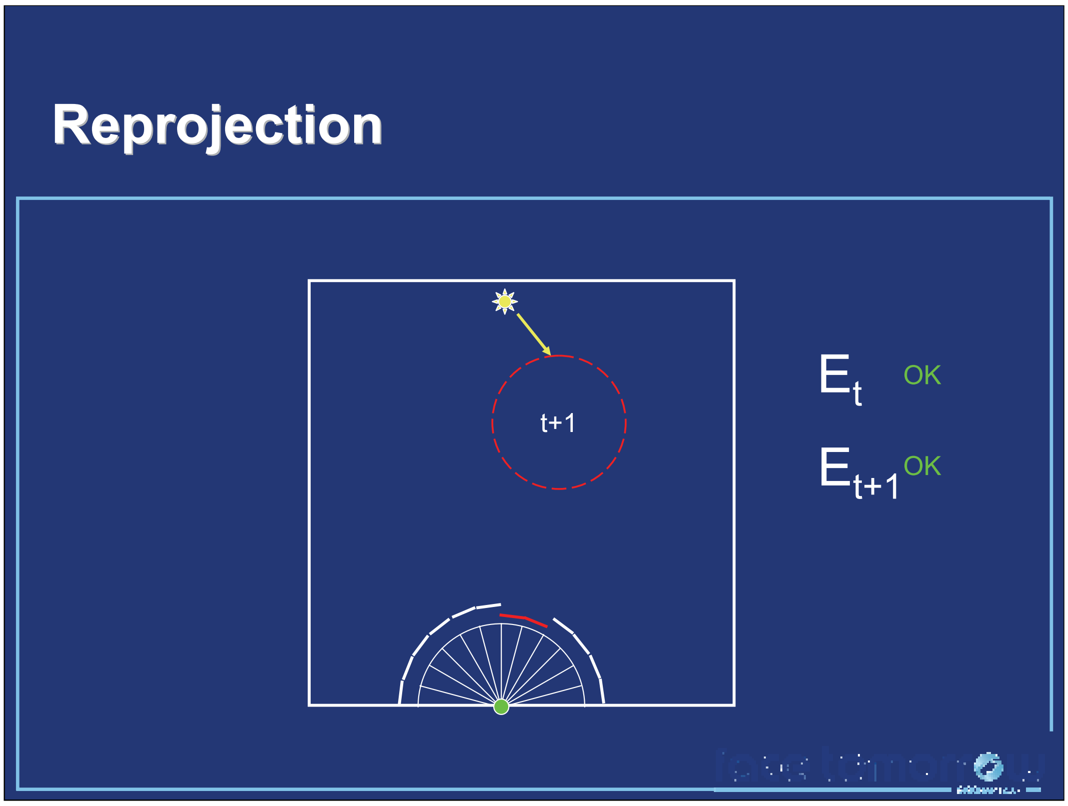
This future position of the red sphere is used to reproject the radiance samples corresponding to the sphere to the new position. This reprojection yields missing information at the former location of the sphere, and overlapping values at the novel position.



Reprojection

A depth culling step chooses the closest value in the case of overlapping radiance values. In this case, the background values are culled.

A simple hole-filling is applied in strata where information is missing. Since we only deal with small movements (1 frame), we simply fill the holes using the closest neighboring value.

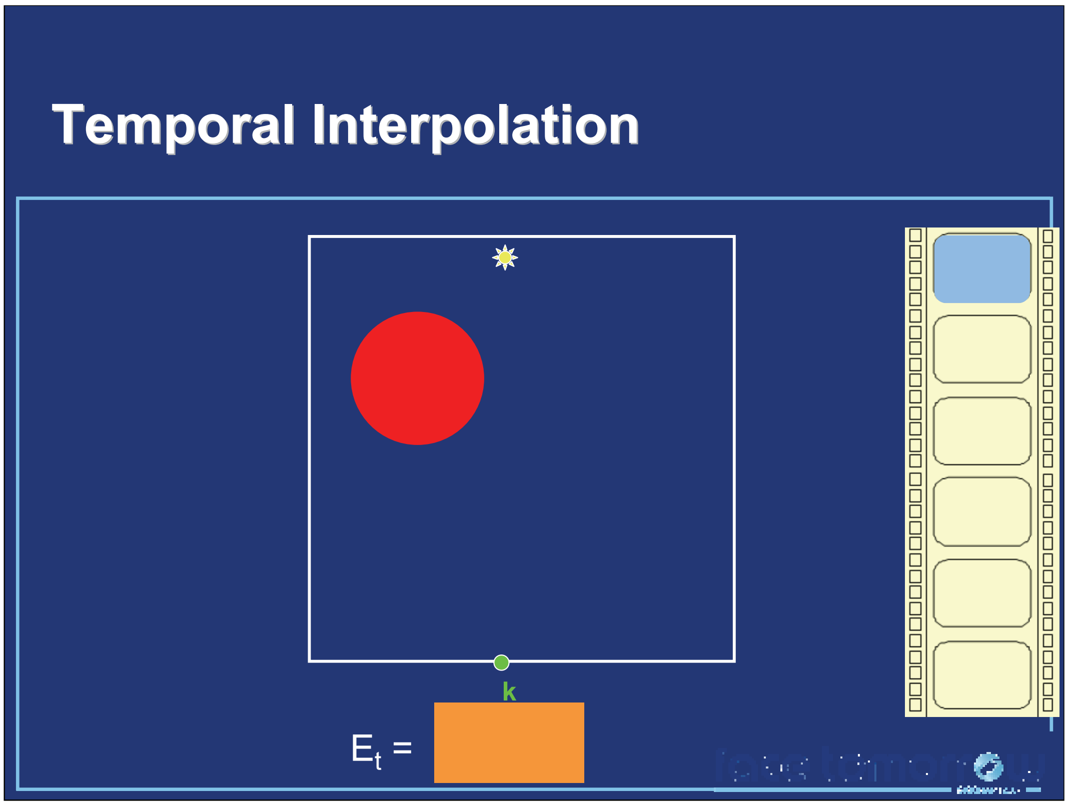


Reprojection

E_t OK

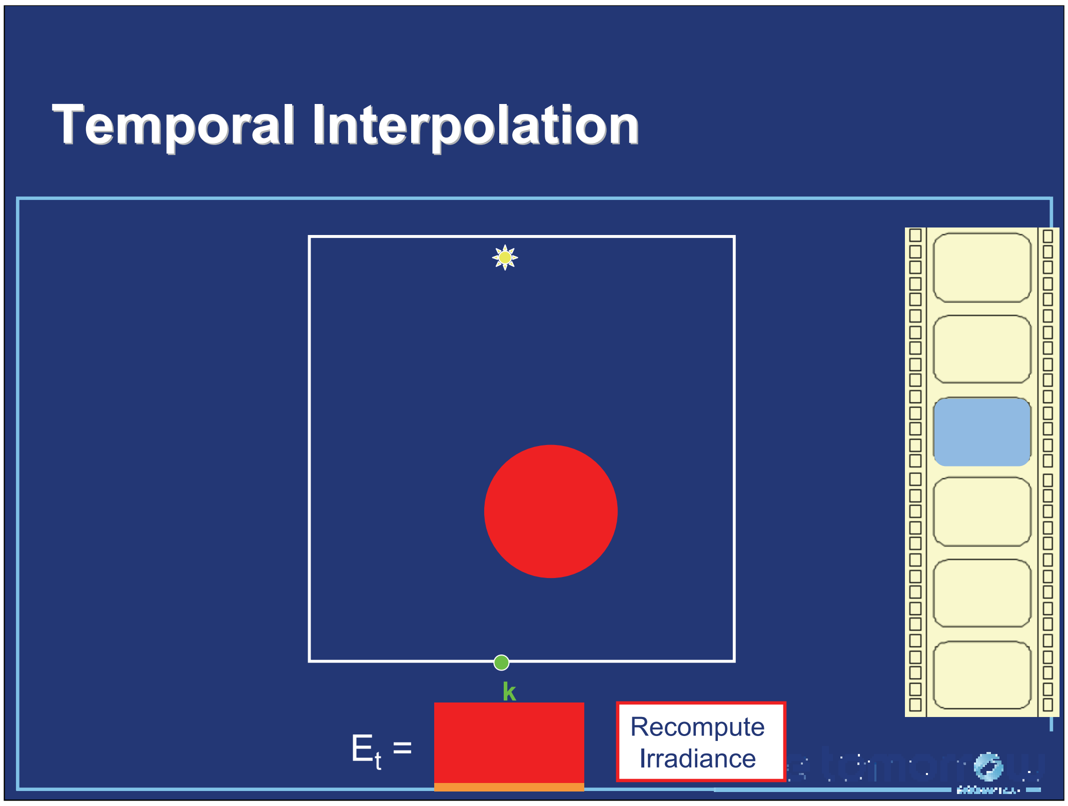
E_{t+1} OK

After depth culling and hole filling, our method provides a reliable estimate of the future incoming lighting. The lifespan of the record created in k is thus completely defined.



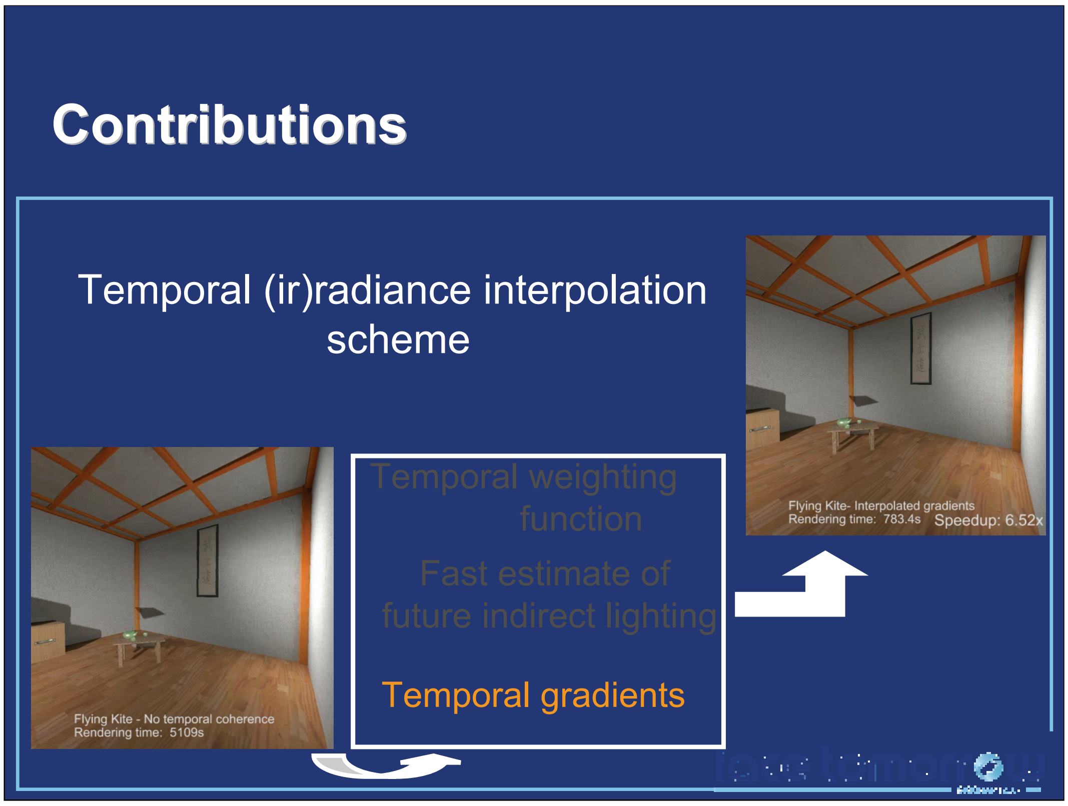
Temporal Interpolation

At this point, we know that the value of record k can be reused across n frames.



Temporal Interpolation

At the end of the record lifespan, a new record is computed, containing an up-to-date irradiance value. In this case the red sphere got closer to k, hence the irradiance at time t and t+n are noticeably different. A consequence of this brutal replacement is sudden changes in the color of image portions, known as « popping » artifacts.



Contributions

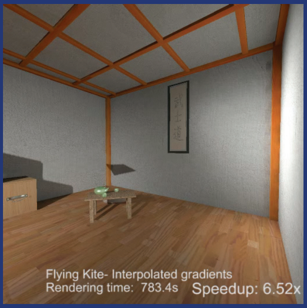
Temporal (ir)radiance interpolation scheme

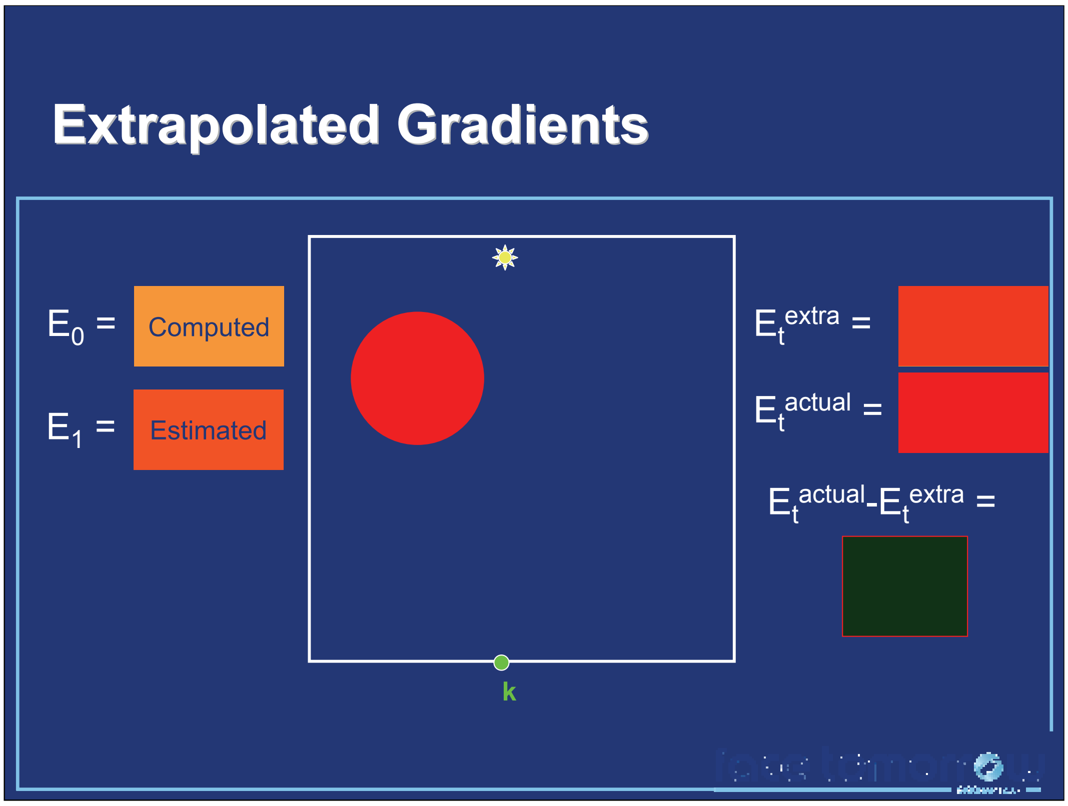
Flying Kite - No temporal coherence
Rendering time: 5109s

Temporal weighting function

Fast estimate of future indirect lighting

Temporal gradients





Extrapolated Gradients

$E_0 =$ Computed

$E_1 =$ Estimated

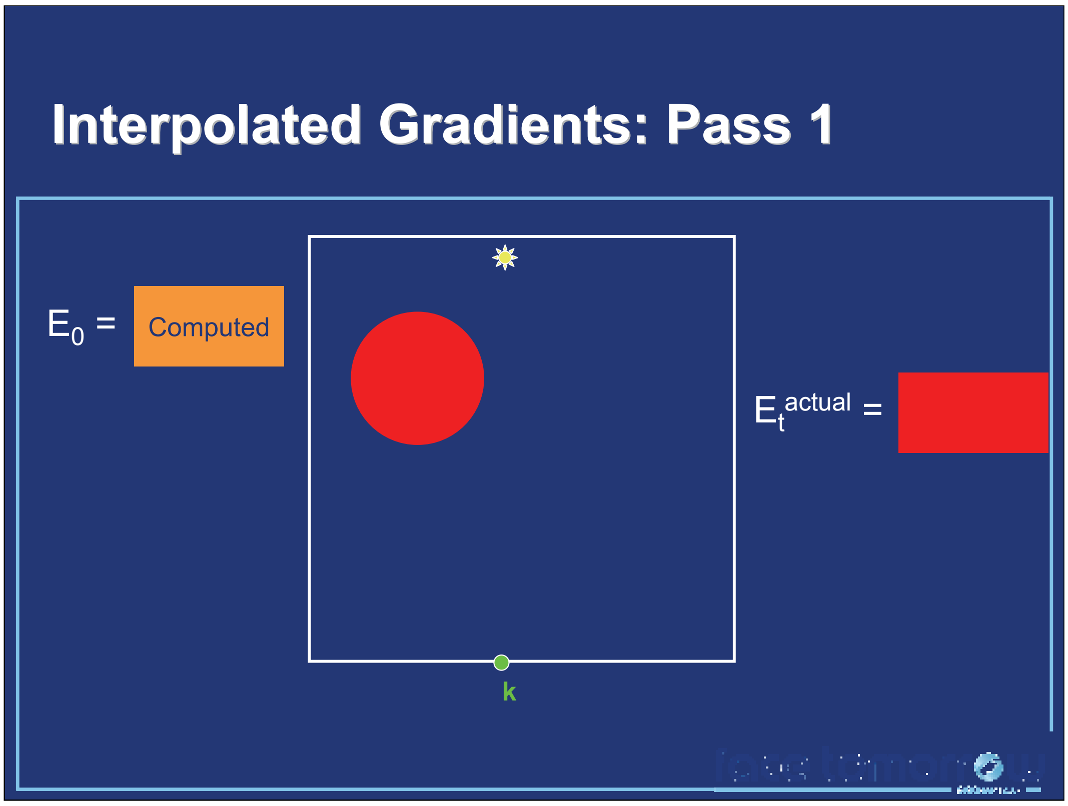
$E_t^{extra} =$

$E_t^{actual} =$

$E_t^{actual} - E_t^{extra} =$

First, we propose to use the estimate of the future incoming lighting to extrapolate the lighting over the entire lifespan of the record. While this method reduces the gap between the extrapolated and the actual irradiance values, the difference is still not negligible. As a consequence, some popping artifacts will remain visible.

This method has one major advantage: the indirect lighting can be computed and displayed on the fly, as the animation is played. Particularly, such gradients could be used in the context of interactive global illumination computation.



Interpolated Gradients: Pass 1

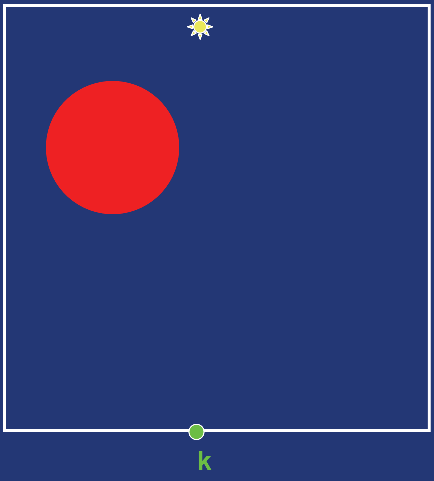
Interpolated gradients completely avoid the popping artifacts by temporally interpolating the irradiance. In a first pass, records are generated as explained previously: each record is used within its lifespan, and new records are computed as the lifespans expire. Let us consider a record R_0 computed at time t_0 and located at point k. When this record expires after n frames, it is replaced by a new record R_1 also located at point k. The temporal gradient of R_0 is then deduced from $(E_{R1}-E_{R0})/n$.

This gradient approximates the change of lighting within the lifespan of a record by linearly interpolating the irradiance at the beginning and at the end of the lifespan. While completely removing popping artifacts, this linear approximation may smooth out high frequency changes that may happen during the lifespan of the record. Accounting for such changes either require a reduction of a^t and δ_{tmax}, or the definition of a higher order interpolation scheme.

Interpolated Gradients: Pass 2

E_0 = Computed

E_t = Computed



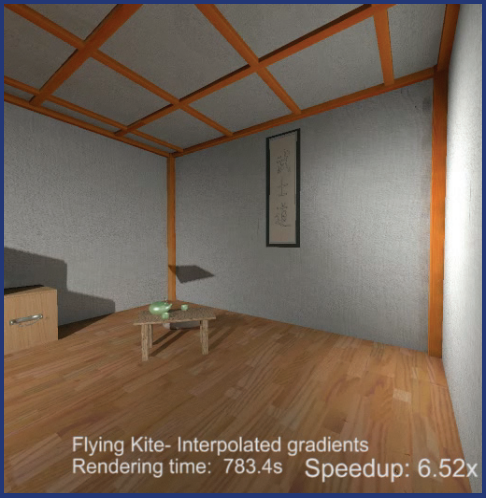
E_t^{inter} =

E_t^{actual} =

$E_t^{actual} - E_t^{inter}$ =

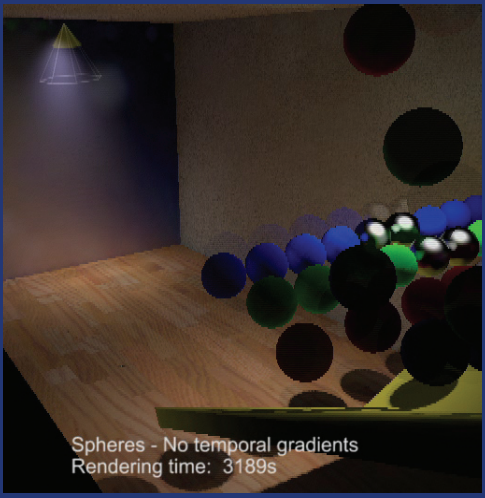
Results: Flying Kite

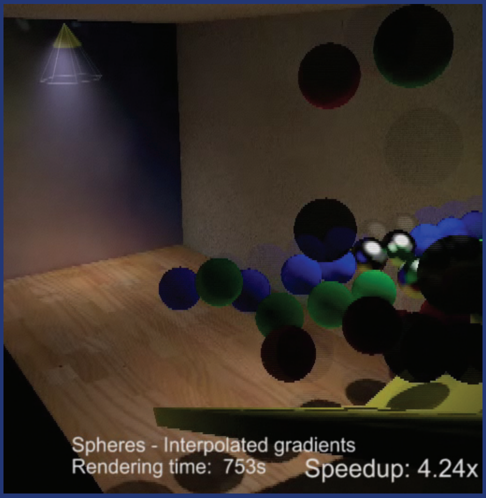
Flying Kite - No temporal coherence
Rendering time: 5109s



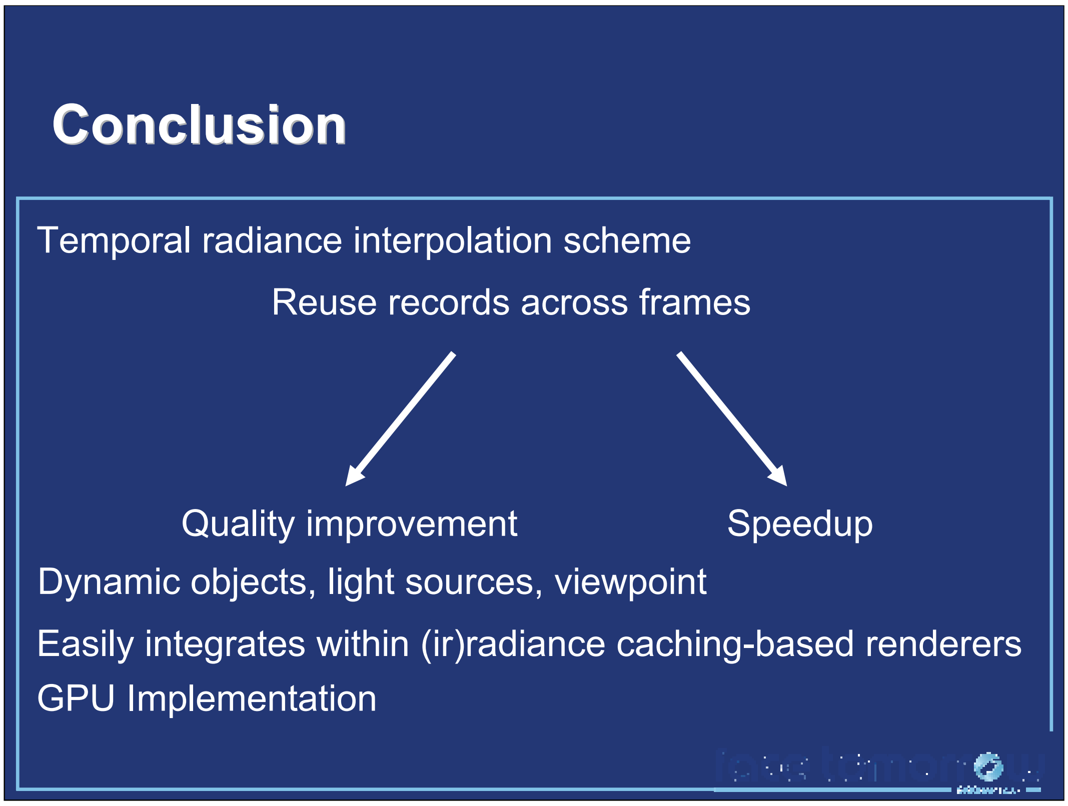
Videos illustrating the interpolated gradients and the adaptive record lifespan in different scenes can be found in [MyWebSite].

Results: Spheres





This method can be straightforwardly extended to nondiffuse interreflexions using radiance caching.



Conclusion

Temporal radiance interpolation scheme

$$\begin{tikzcd}
 & \text{Reuse records across frames} \arrow[dl] \arrow[dr] & \\
\text{Quality improvement} & & \text{Speedup}
\end{tikzcd}$$

Dynamic objects, light sources, viewpoint

Easily integrates within (ir)radiance caching-based renderers

GPU Implementation

The method described in this part is based on the reuse of irradiance records across frames. While reducing the computational cost of animation rendering, the flickering artifacts are drastically reduced. This method supports any type of dynamic scene components, and can be easily integrated in existing renderers.

Future work will consider the elimination of the maximum lifespan δ_{tmax}. Also, we would like to devise methods for higher order temporal interpolation to account for sharp changes of indirect lighting.

Bibliography

[Foley05] Foley T., Sugerman J. “KD-tree acceleration structures for a GPU raytracer”, 2005

[Krivanek05a] Krivanek J., Gautron P., Pattanaik S., Bouatouch K. “Radiance Caching for Efficient Global Illumination Computation”, 2005

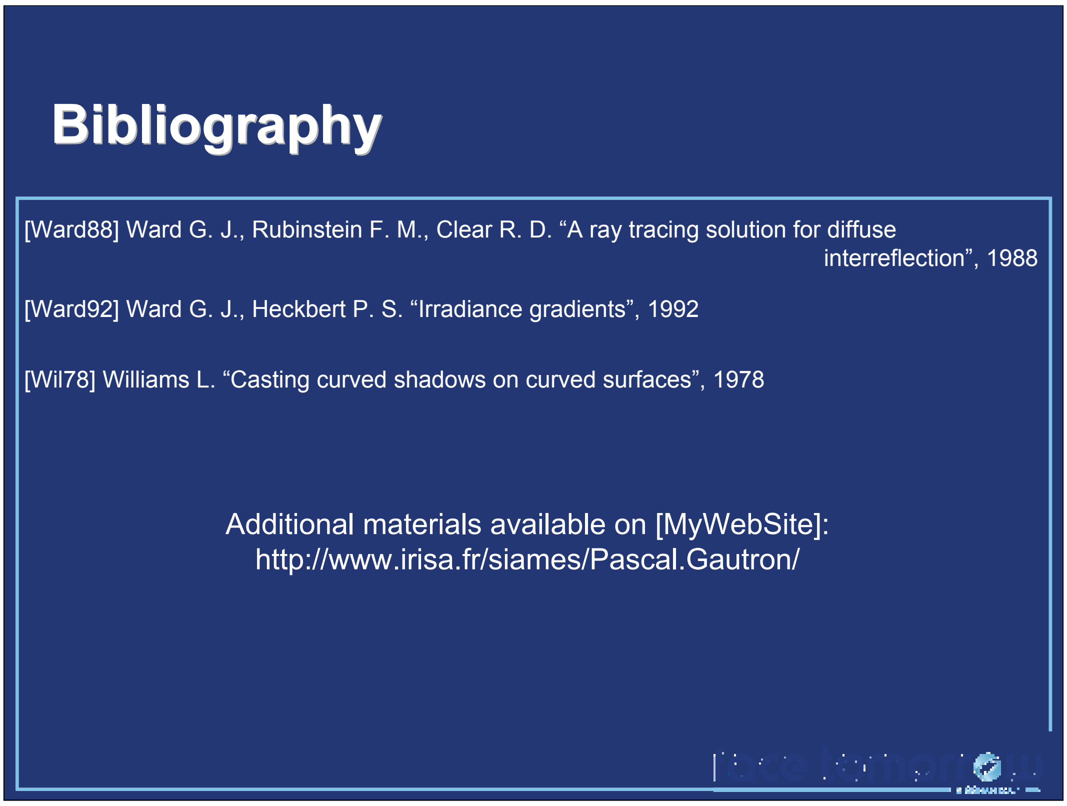
[Krivanek05b] Krivanek J., Gautron P., Bouatouch K., Pattanaik S. “Improved Radiance Gradient Computation”, 2005

[LC04] Larsen B. D., Christensen N. “Simulating photon mapping for real-time applications”, 2004

[Purcell02] Purcell T. J., Buck I., Mark W. R., Hanrahan P. “Ray tracing on programmable graphics hardware”, 2002

[Purcell03] Purcell T. J., Donner C., Cammarano M., Jensen H. W., Hanrahan P. “Photon mapping on programmable graphics hardware”, 2003

[Walter99] Walter B., Drettakis G., Parker S. “Interactive rendering using the render cache”, 1999



Bibliography

[Ward88] Ward G. J., Rubinstein F. M., Clear R. D. “A ray tracing solution for diffuse interreflection”, 1988

[Ward92] Ward G. J., Heckbert P. S. “Irradiance gradients”, 1992

[Wil78] Williams L. “Casting curved shadows on curved surfaces”, 1978

Additional materials available on [MyWebSite]:
http://www.irisa.fr/siames/Pascal.Gautron/