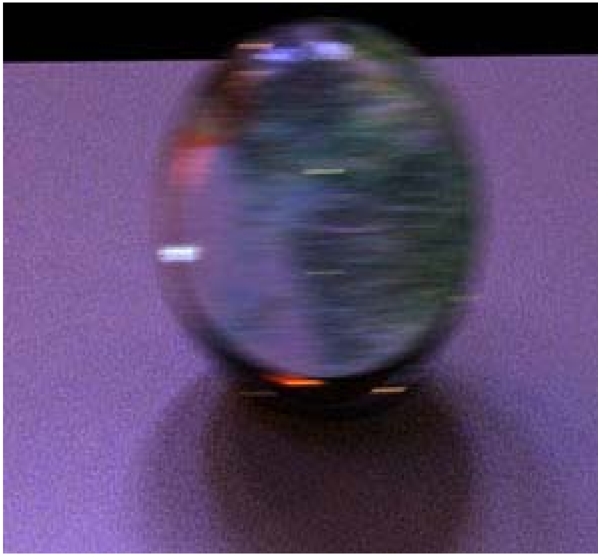
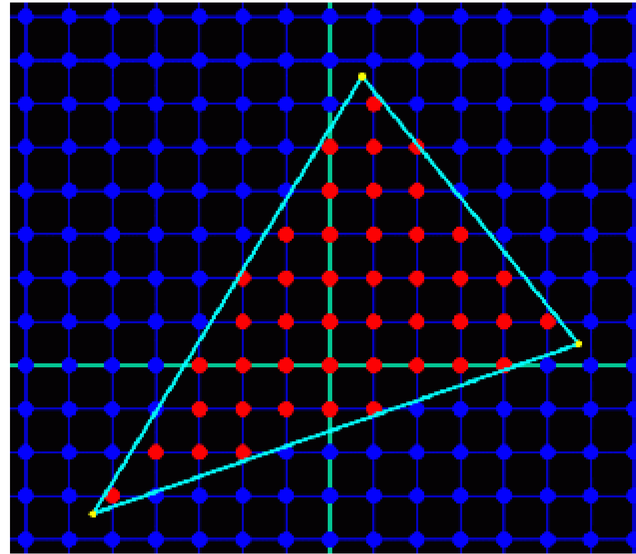
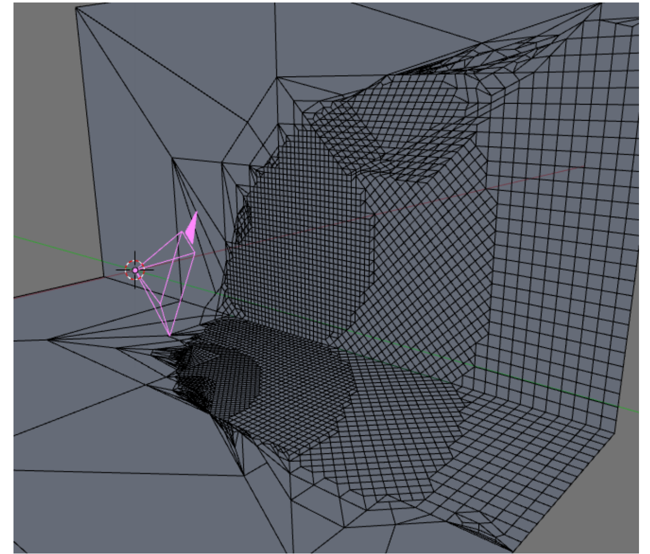






IFT3355

INFOGRAPHIE

SECTION 12: SYSTÈMES DE RENDU

Derek Nowrouzezahrai et Pierre Poulin

Département d'informatique et de recherche opérationnelle

Université de Montréal

Université de Montréal

Systèmes de rendu: ça sert à quoi?

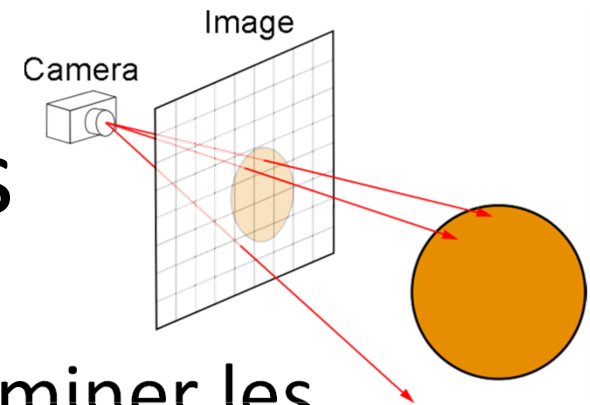
- Les trois systèmes que nous étudierons ont été toutes conçus pour résoudre le même problème:

la détermination des surfaces visibles par respect de l'œil (*visible surface determination)*

- Chacun a également évolué afin de supporter des effets beaucoup plus complexes (ex: GI)
  - En fait, les limitations structurels et/ou inconvénients présents dans chacun de ces architectures peuvent être typiquement attribués au fait que l'ombrage / l'illumination a seulement été considérer bien après la conception originale de chaque architecture

Université de Montréal

Famille 1: Lancer de rayons



- Originalement conçu pour déterminer les surfaces visibles de l'œil (caméra *pinhole*)

```
for( chaque pixel p )
    Générer un rayon r de l'oeil à p

    float dist_min = infinité
    Intersection h = null

    for( chaque objet o )
      if( trace( r, o ) et hit_dist <= dist_min )
        dist_min = hit_dist
        h.objet = o et h.dist = hit_dist

    p.couleur = shade(h)
```

Famille 1: Lancer de rayons

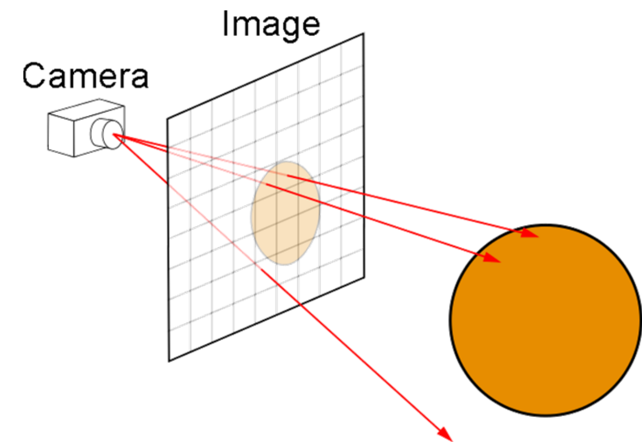
Visibilité:

1. Générer les rayons de l'oeil
2. Coder un fonction **`trace`** pour
   - des plans
   - des sphères
   - des triangles, etc.
3. Trouver l'intersection **la plus proche**

Illumination (locale, globale):

4. Qu'est-ce qu'on peut accomplir avec **`shade`**?

```
for( chaque pixel p )
  Générer rayon r de l'oeil à p
  float dist_min = infinité
  intersection h = null
  for( chaque objet o )
    if( trace( r, o ) et
          hit_dist <= dist_min )
        dist_min = hit_dist
        h.objet = o
        h.dist = hit_dist
p.couleur = shade(h)
```

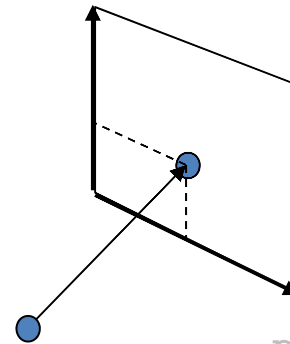


Université de Montréal

La forme générale d'un rayon

```
for( chaque pixel p )
  Générer rayon r de l'oeil à p
  float dist_min = infinité
  intersection h = null
  for( chaque objet o )
    if( trace( r, o ) et
        hit_dist <= dist_min )
      dist_min = hit_dist
      h.objet = o
      h.dist = hit_dist
p.couleur = shade(h)
```

- Un rayon est défini par
  - une origine (x_0, y_0, z_0)
  - une direction $(\Delta x, \Delta y, \Delta z)$



- Pour les rayons de l'oeil:
  - défini par la position du caméra et un point sur la fenêtre (x_1, y_1, z_1)
- Sous forme paramétrique un rayon s'exprime comme

$$P(t) = P_0 + t(P_1 - P_0)$$

$$x = x_0 + t\Delta x \qquad y = y_0 + t\Delta y \qquad z = z_0 + t\Delta z$$

$$\Delta x = x_1 - x_0 \quad \Delta y = y_1 - y_0 \quad \Delta z = z_1 - z_0$$

Université de Montréal

Intersection avec un plan

- L'intersection d'un rayon avec un plan (*A,B,C,D*) consiste à exprimer *les points sur le plan* en fonction des points sur le rayon qui nous intéresse

$$Ax + By + Cz + D = 0$$

$$A(x_0 + t\Delta x) + B(y_0 + t\Delta y) + C(z_0 + t\Delta z) + D = 0$$

$$t = \frac{-(Ax_0 + By_0 + Cz_0 + D)}{A\Delta x + B\Delta y + C\Delta z}$$

- Pour l'intersection avec un polygone, il faudra déterminer si le point d'intersection est à l'intérieur du polygone

```
for( chaque pixel p )
  Générer rayon r de l'oeil à p
  float dist_min = infinité
  intersection h = null
  for( chaque objet o )
     if( trace( r, o ) et
         hit_dist <= dist_min )
       dist_min = hit_dist
       h.objet = o
       h.dist = hit_dist
p.couleur = shade(h)
```

Université de Montréal

Intersection avec une sphère

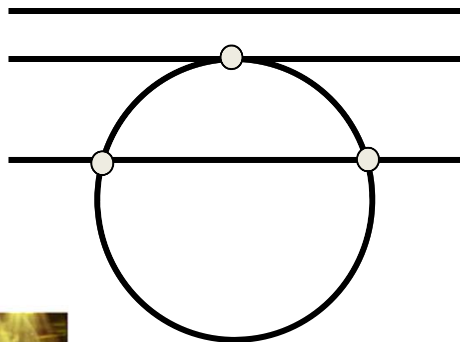
- Similairement pour l'intersection d'un rayon avec une sphère

$$x^2 + y^2 + z^2 = 1$$

$$(x_0 + t\Delta x)^2 + (y_0 + t\Delta y)^2 + (z_0 + t\Delta z)^2 = 1$$

$$(\Delta x^2 + \Delta y^2 + \Delta z^2)t^2 + 2(x_0\Delta x + y_0\Delta y + z_0\Delta z)t + (x_0^2 + y_0^2 + z_0^2 - 1) = 0$$

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$



0, 1 ou 2 intersections réelles

Intersection en espace objet

Mr

M^{-1}

M

r

n

q

$M^{-1}b$

$M^{-1}a$

$M^{-1}a + tM^{-1}b$

Mq

$(M^{-1})^T n$

espace monde

espace objet

b

$a + tb$

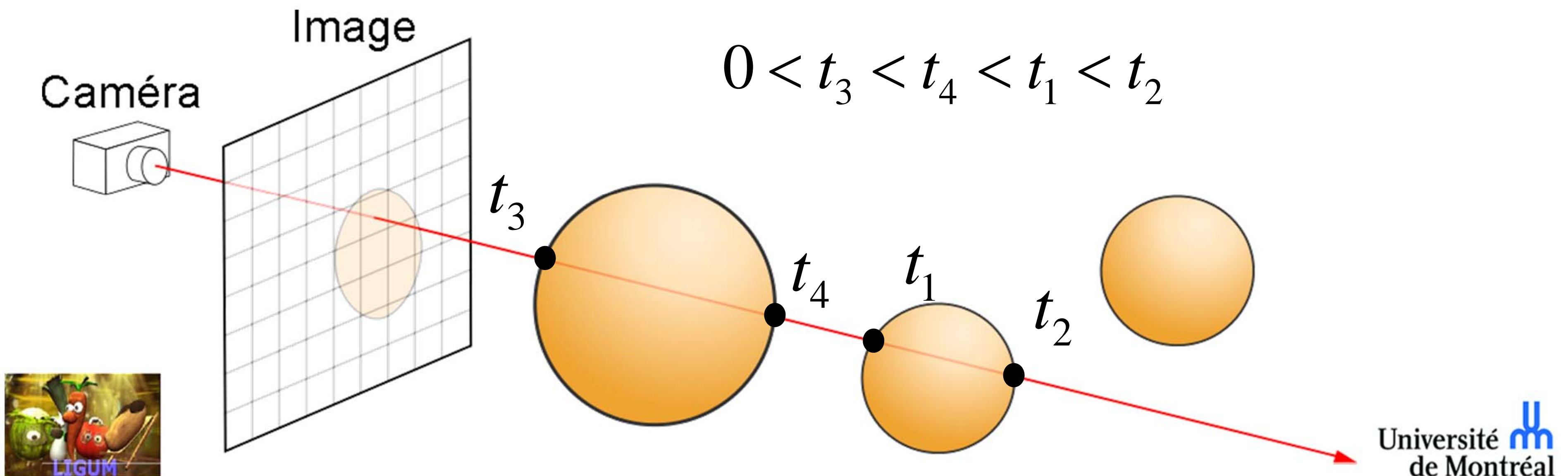
a

Université de Montréal

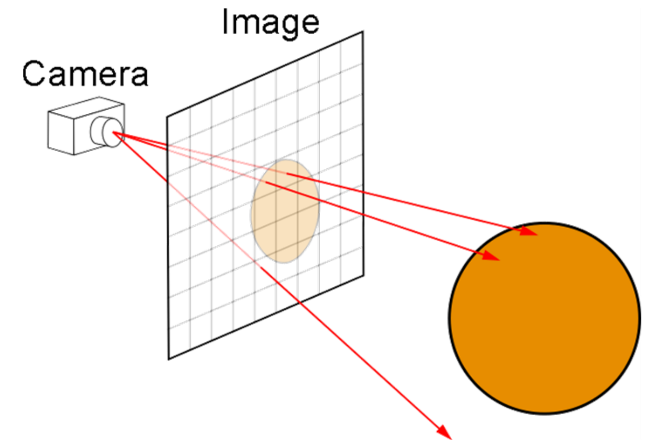
Famille 1: Lancer de rayons

- Pour identifier la surface visible du pixel/caméra, il suffit d'intersecter *tous* les objets et de ne conserver que l'intersection ayant la plus petite valeur positive de t

```
for( chaque pixel p )
  Générer rayon r de l'oeil à p
  float dist_min = infinité
  intersection h = null
  for( chaque objet o )
     if( trace( r, o ) et
         hit_dist <= dist_min )
       dist_min = hit_dist
       h.objet = o
       h.dist = hit_dist
p.couleur = shade(h)
```



Famille 1: Lancer de rayons

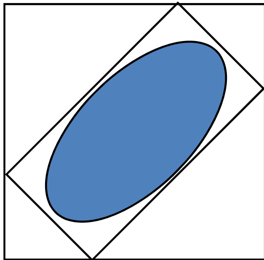


- Compléxité
  - *brute-force* en temps: O(p n)
  - avec un structure d'accélération: O(p log n)
    - Noter qu'un tel structure aura besoin de O(n log n) de temps pour sa construction!
    - Noter aussi que chaque représentation supplémentaire de ta scène encourt un coût en mémoire!

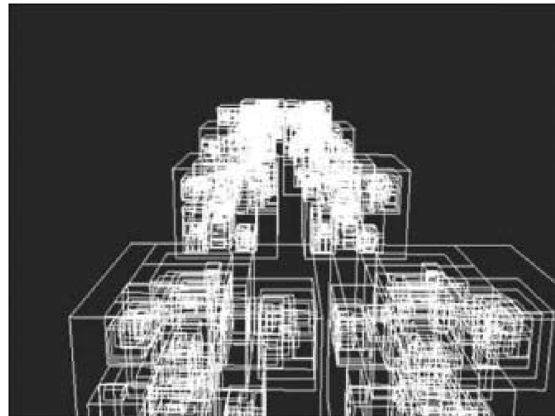
Université de Montréal

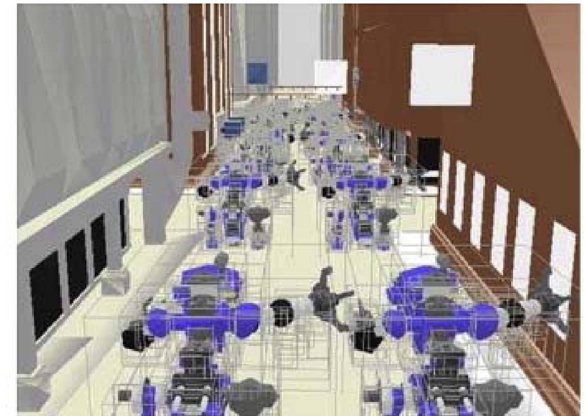
Volumes englobants

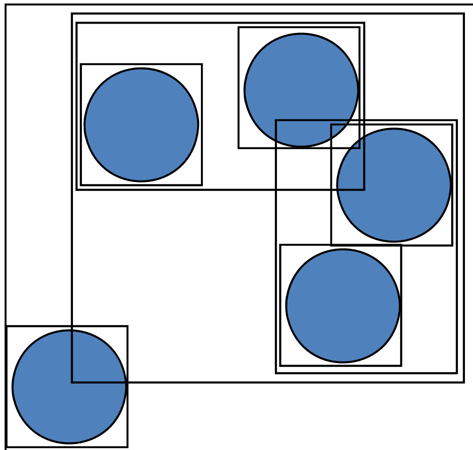
boîte orientée

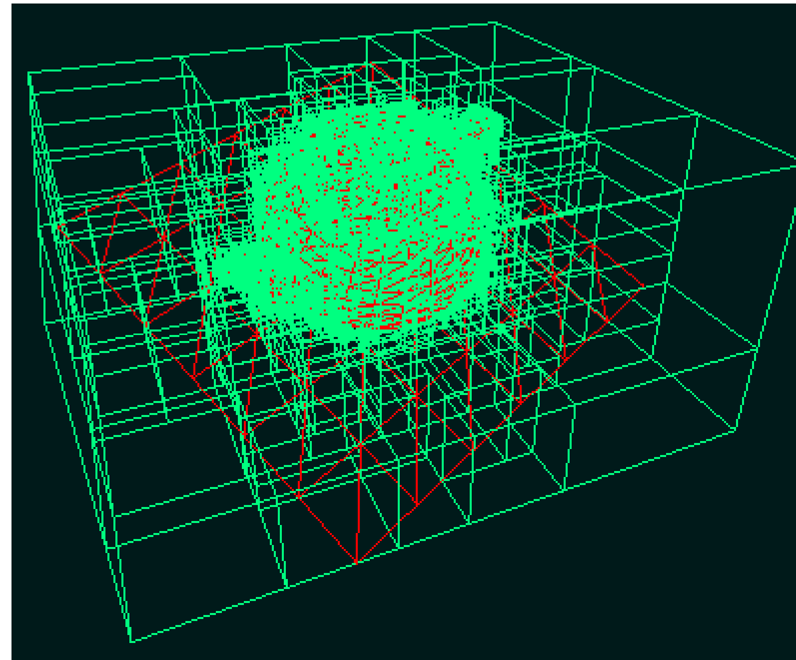


boîte alignée sur les axes







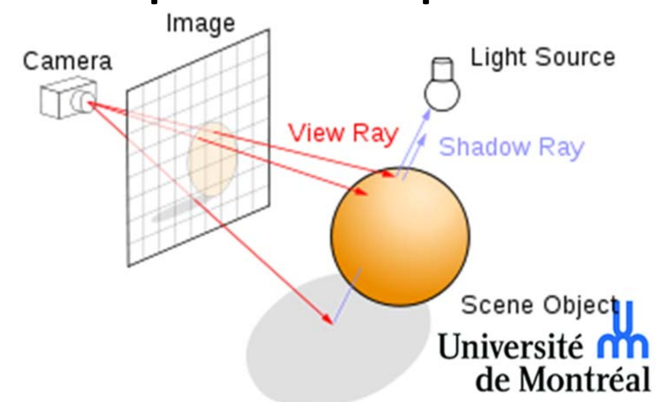


hiérarchie de boîtes englobantes (*BVH*)

Université de Montréal

Famille 1: Lancer de rayons

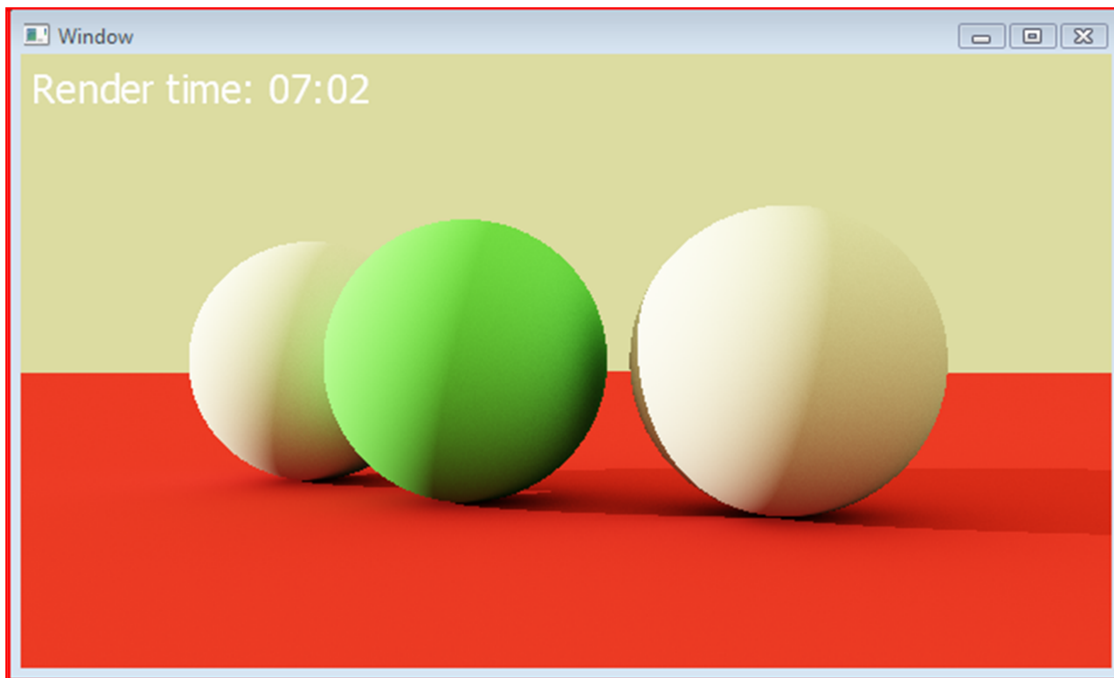
- Les lanceurs de rayons ont graduellement été étendus afin de permettre la simulation des effets de plus en plus complexes comme:
  - Les ombres dur et les réflexions miroir parfaits
  - Le flou de mouvement (*motion blur*)
  - Profondeur de champ (*depth of field*)
  - Simulation de modèle d'illumination plus complexes
  - L'illumination globale



Université de Montréal

Famille 1: Lancer de rayons

- Presque toutes ces extensions se réalise complètement en dehors de l'algorithme de base




```
for( chaque pixel p )
  Générer rayon r de l'oeil à p
  float dist_min = infinité
  intersection h = null
  for( chaque objet o )
    if( trace( r, o ) et
        hit_dist <= dist_min )
      dist_min = hit_dist
      h.objet = o
      h.dist = hit_dist
p.couleur = shade(h)
```

Université de Montréal

Famille 1: Lancer de rayons

- Implémentation HW
  - Deux problèmes avec les lanceurs de rayons qui rends les implémentations HW difficiles:
    - Cohérence des rayons secondaires, et
    - Accès globale de la scène
- Construire et traverser un structure d'accélération sur ex: un GPU d'un manière efficace est très difficile

High Performance Graphics (2010)
M. Doggett, S. Laine, and W. Hunt (Editors)

Architecture Considerations for Tracing Incoherent Rays

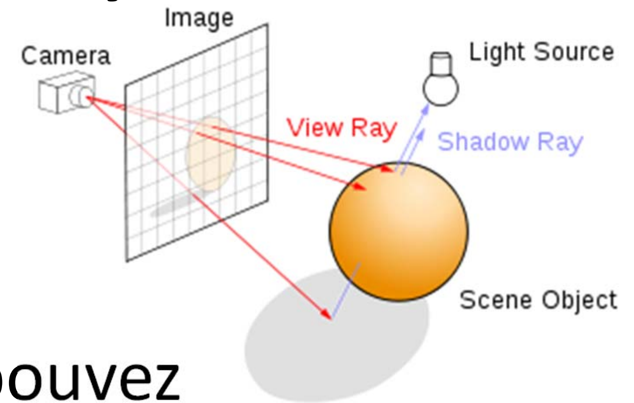
Timo Aila Tero Karras
NVIDIA Research

Understanding the Efficiency of Ray Traversal on GPUs

Timo Aila* Samuli Laine*
NVIDIA Research NVIDIA Research

Université de Montréal

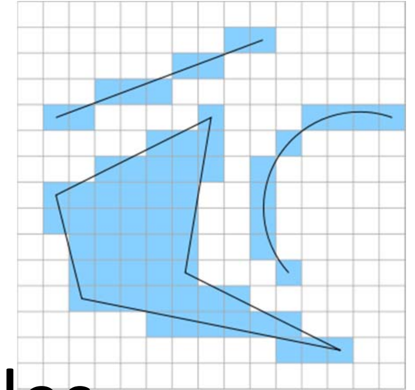
Famille 1: Lancer de rayons



- Avantages:
  - L’algorithme le plus flexible: vous pouvez implémenter n’importe quel algorithme de rendu *précisément* dans un lanceur de rayons
- Désavantages:
  - Typiquement le plus lent des algorithmes
  - Les structures d’accélération prennent typiquement plus d’espace que la scène originale! Difficile (impossible?) à coder comme moteur de *streaming*

Université de Montréal

Famille 2: Rastérisation



- Similairement conçu pour déterminer les surfaces visibles de l'œil (caméra *pinhole*)
  - Supposition additionnel: que chaque objet est beaucoup plus grand qu'un pixel

```
for( chaque objet o )
    Projeter o sur le plan d'image

    for( chaque pixel p )
      if( isinside(p, o) )
        // effectuer un test de z-buffer
        if( o.z < zbuffer[p] )
          zbuffer[p] = o.z
          p.couleur = shade(o)
```

Famille 2: Rastérisation

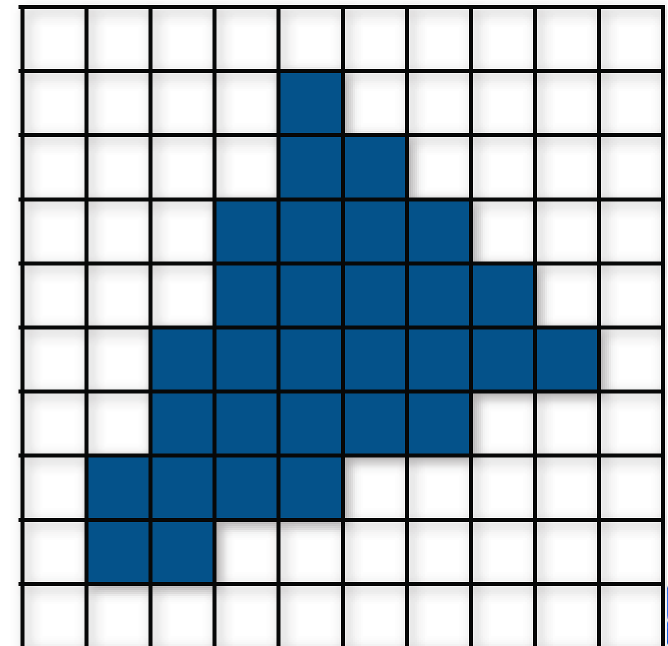
<u>Visibilité:</u>

1. Transformer et <u>projeter</u> chaque objet (triangle)
2. Coder un fonction **`isinside`** pour
   - des triangles
   - des polygons, etc.
3. Trouver l'objet **la plus proche** avec un *z-buffer*

<u>Illumination (locale, globale):</u>

4. Calculer le couleur du pixel avec **`shade`**

```
for( chaque objet o )
    Projeter o sur l'image
    for( chaque pixel p )
      if( p est dedans o )
        // test de z-buffer
        if( o.z < zbuffer[p] )
          zbuffer[p] = o.z
          p.couleur = shade(o)
```

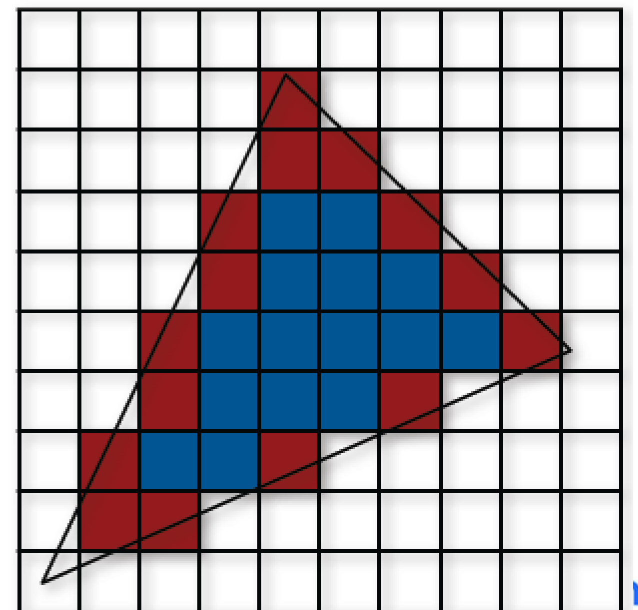


Rastérisation – remplissage: *scan conversion* (séquentielle)

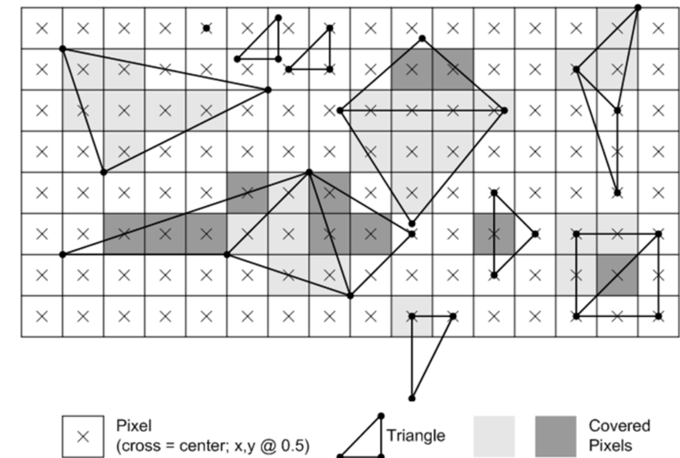
- Après les transformations et projections (selon les matrices standards), la remplissage des triangles est l'opération le plus important d'un moteur de rastérisation

```
for( chaque objet o )
   Projeter o sur l'image
   for( chaque pixel p )
     if( p est dedans o )
       // test de z-buffer
       if( o.z < zbuffer[p] )
         zbuffer[p] = o.z
         p.couleur = shade(o)
```

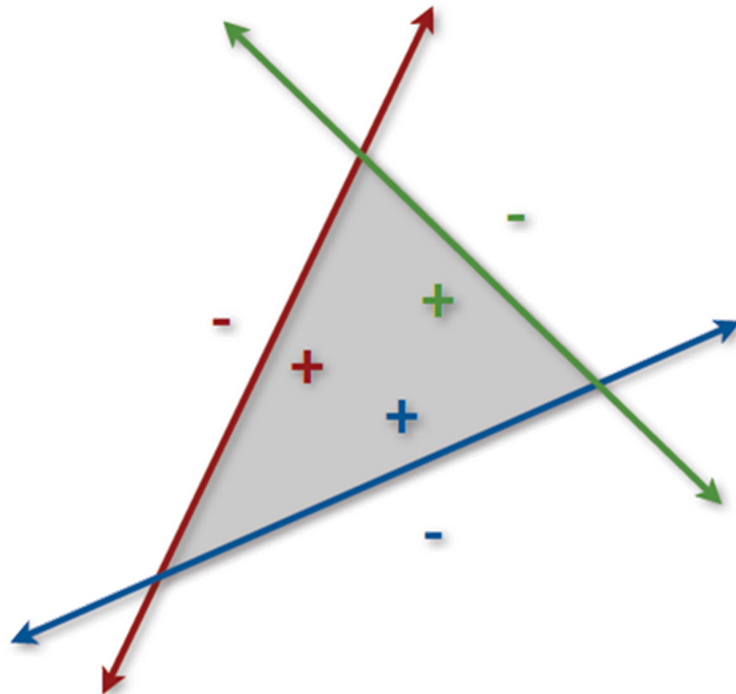
- Une méthode de remplissage qui procède d'un manière séquentielle est le *scan conversion:*
    1. Commence en traversant chaque ranger de pixel (un *scan line*)
    2. Garder trace des arrêts/bords
    3. Remplit les trames de pixels



Rastérisation – remplissage: y-a-t'il un *isinside* test parallèle?



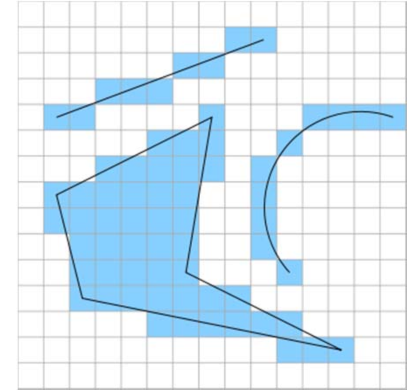
- Il y a plusieurs manières de rapidement déterminer si un pixel est dedans un triangle (en parallèle)
  - Test de demi-espace (*half-space tests*)
  - Utilisation des coordonnées Barycentrique
- Fait attention autour des arrêts!




```
for( chaque objet o )
    Projeter o sur l'image
    for( chaque pixel p )
      if( p est dedans o )
        // test de z-buffer
        if( o.z < zbuffer[p] )
          zbuffer[p] = o.z
          p.couleur = shade(o)
```

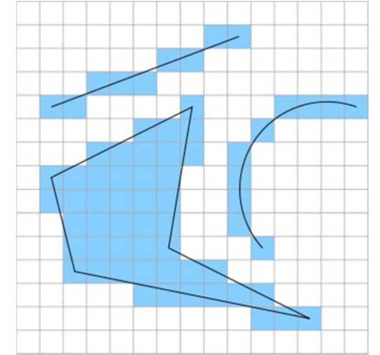
Université de Montréal

Famille 2: Rastérisation



- Complexité
  - *Brute-force* en temps: O(n p)
  - Nous pouvons encore utiliser des structures d'accé-lération pour améliorer la complexité à O(p log n)
    - *Z-buffer* hiérarchique
    - Boites englobantes (projetées): pas nécessaire de tester tous les pixels pour chaque objet pendant le remplissage
    - Pour les scènes statiques, beaucoup d'optimisation possible (ex: *PVS*)
  - La rastérisation traite la géométrie *projetée*, où la détermination de visibilité a lieu en espace image
    - Ces opérations sont beaucoup plus facile à paralléliser

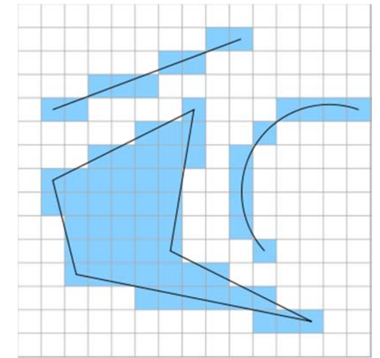
Famille 2: Rastérisation



- Comme avec lancer de rayons, le modèle de rastérisation est toujours en évolution
- Des centaines d'extensions existe pour permettre la simulation des effets plus complexe dans le cadre d'un système de rastérisation:
  - Les ombres dur et étendues
  - L'illumination globale
- Un développement récent s'appelle la rastérisation stochastique qui cible la simulation des effets de distributions

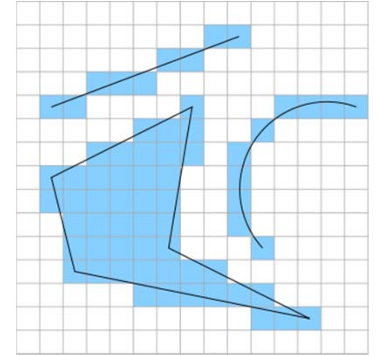
Université de Montréal

Famille 2: Rastérisation



- Le structure de rastérisation (*streaming-friendly,* opérations en espace image, transformations appliqués indépendamment à chaque objet, etc.) rend son implémentation en HW très facile
- OpenGL et DirectX sont des API modelés autour d'un concept de rastérisation extensible et réalisable sur des GPUs spécialisés

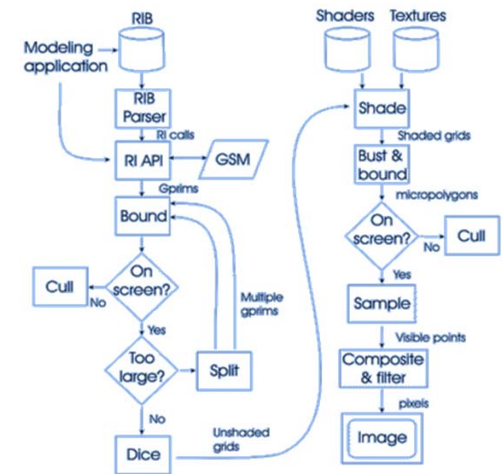
Famille 2: Rastérisation



- Avantages:
  - *Stream-friendly*
  - Facile à paralléliser
- Désavantages
  - Aucun accès directe à des informations *globale*
  - Même l'implémentation des effets simples peut nécessiter des solutions "créatives"
  - Très difficile à simuler des effets d'illumination avancés (où la cohérence en espace rayons est réduite)

Université de Montréal

Famille 3: Micropolygon



- Quand les objets (projetés) sont plus petit que la taille d'un pixel, on tombe sur un cas particulier:
  - Premièrement, beaucoup des problèmes de visibilité (de l'œil) deviennent plus facile à gérer
  - Deuxièmement, la rastérisation ne marche plus
  - L'interpolation barycentrique entre espace objet est espace image n'est plus nécessaire
- Il existe une troisième famille de système conçu avec cette scénario comme cas de base

Famille 3: Micropolygon

- Contrairement à lancer de rayons et rastérisation, le système micropolygon de **REYES** a été conçu non seulement pour calculer la visibilité de l'œil, mais aussi pour:
  - Gérer des scènes très complexes et avec beaucoup de détails géométriques
  - Supporter quelques effets de shading avancés qui ont été identifiés comme important pour le réalisme: flou de mouvement et *displacement*
  - Première système de shaders programmable

Famille 3: Micropolygon

- L’algorithme de REYES (une exemple de rastérisation de micropolygon) est malheureusement plus complexes qu’un système de lancer de rayons ou de rastérisation
  – Mais il est heureusement similaire à la rastérisation

```
for( chaque objet o )
   // Convertir objet en micropolygon
   o’ = BoundSplit(o);
   mp = Dice(o’);

   Shade(mp);
   SampleHide(mp);
```

  – Les premières étapes ont comme bût de convertir chaque objet dans la scène en représentation commun, des micropolygons, avant le calcul de shading et de visibilité

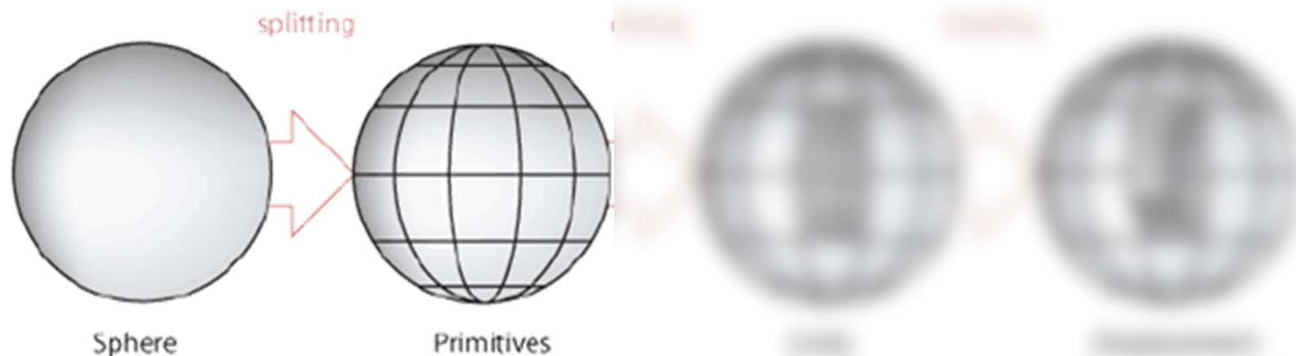
Université de Montréal

Famille 3: Micropolygon

- L’étape **`Bound`** détermine le volume englobant de l’objet et divise l’objet récursivement jusqu’il atteins une taille (projeté) spécifiée en espace image
  - Plusieurs méthodes. Ex: *coarse dicing*, projection des sous-boites englobantes, …
  - Doit prendre en compte les *displacements* potentiels!

```
for( chaque objet o )
   o' = BoundSplit(o);
   mp = Dice(o');

   Shade(mp);
   SampleHide(mp);
```



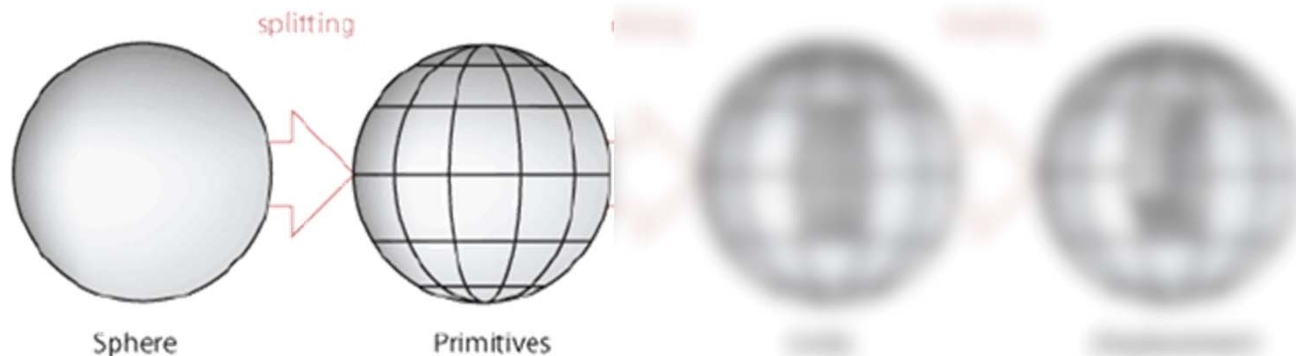
Université de Montréal

Famille 3: Micropolygon

```
for( chaque objet o )
  o' = BoundSplit(o);
  mp = Dice(o');

  Shade(mp);
  SampleHide(mp);
```

- L'étape **`Split`** permet de sous-divisé des objets afin de jeter des parties qui tomberont en dehors le pyramide de vues
  - Similaire aux algorithmes de clipping en rastérisation



Université de Montréal

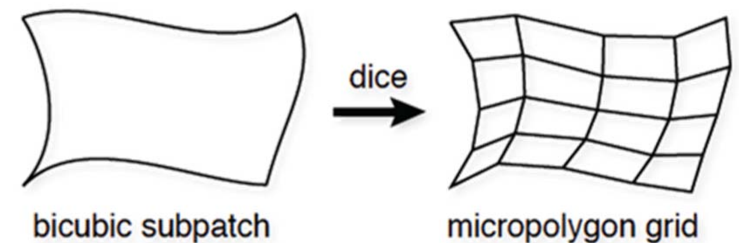
Famille 3: Micropolygon

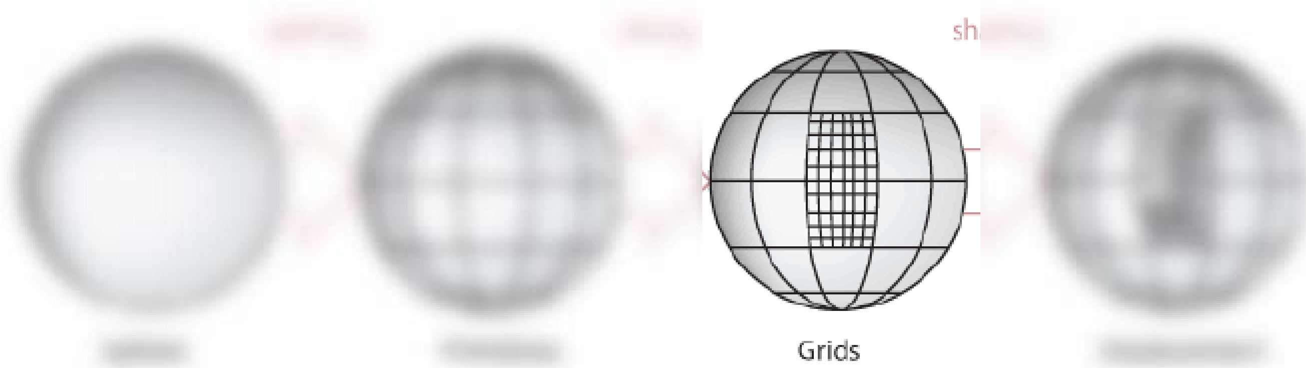
- L'étape **`Dice`** prends les primitives sous-divisés et les utilises pour créer des micropolygons avec des tailles d'environ un pixel

```
for( chaque objet o )
   o' = BoundSplit(o);
   mp = Dice(o');

   Shade(mp);
   SampleHide(mp);
```

- Ces micropolygons sont charger dans un tampons appeler le *microgrid*, chacun avec leurs normals, position, et d'autres informations géométriques





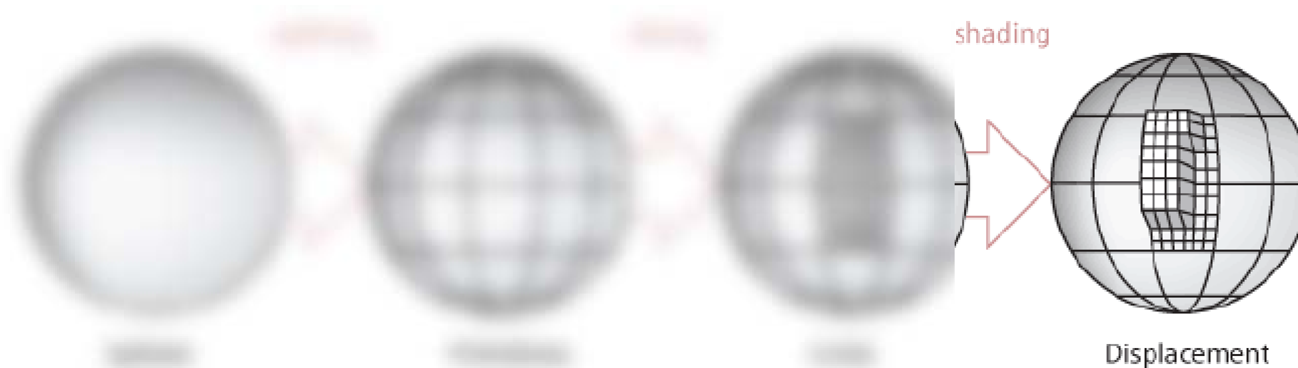
Université de Montréal

Famille 3: Micropolygon

- L'étape **`Shade`** calcule le *displacement* et le *shading* à chaque sommet des micropolygons selon des shaders procédurales
- Tous les propriétés géométriques sont disponibles à une résolution sous-pixel
- Cela peut-être facilement parallélisé

```
for( chaque objet o )
  o' = BoundSplit(o);
  mp = Dice(o');

  Shade(mp);
  SampleHide(mp);
```



Université de Montréal

Famille 3: Micropolygon

- L'étape **`SampleHide`** affiche les micropolygons sur l'image finale, après l'application d'un opération de *busting*, tout en éliminant les surfaces cachés
  - Ici les algorithmes de rastérisation et/ou *z-buffer* peuvent être modifiés afin de déterminer la visibilité de l'œil
- À noter: la complexité du shading est découplé de la complexité de la détermination de visibilité

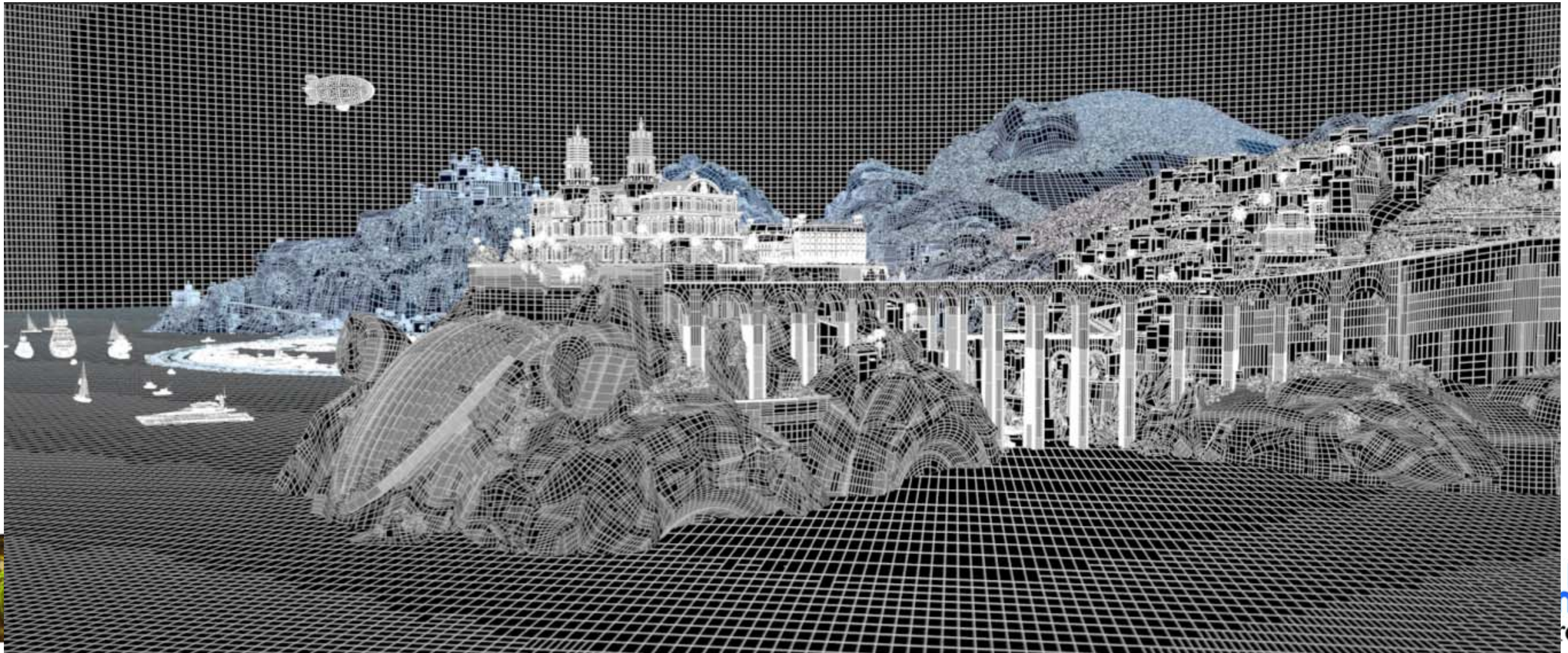
```
for( chaque objet o )
  o' = BoundSplit(o);
  mp = Dice(o');

  Shade(mp);
  SampleHide(mp);
```

Université de Montréal

Famille 3: Micropolygon

- Récemment plusieurs chercheurs ont investigué les manières d'implémenter un système REYES sur les GPUs, et/ou comment étendre les systèmes de rastérisation accélérés pour supporter les scènes plus complexes avec les effets de rendu avancés

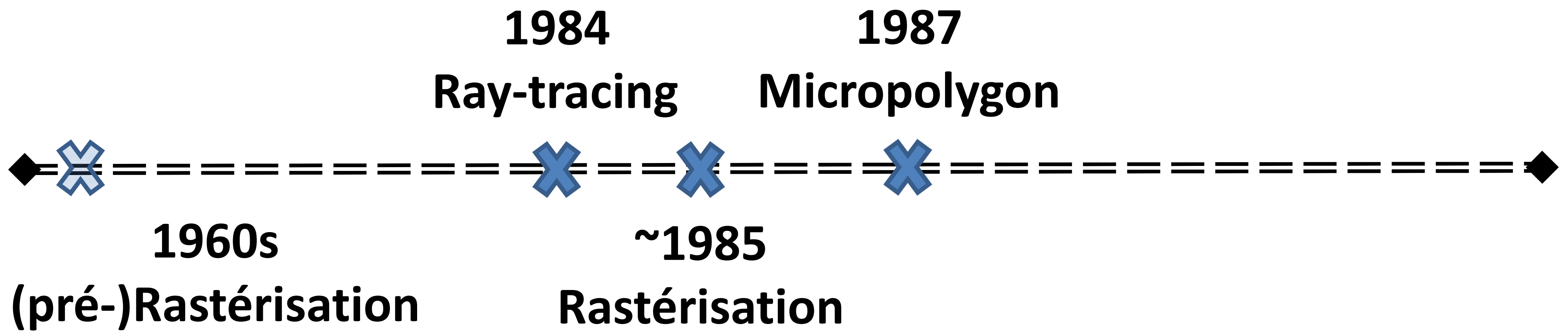


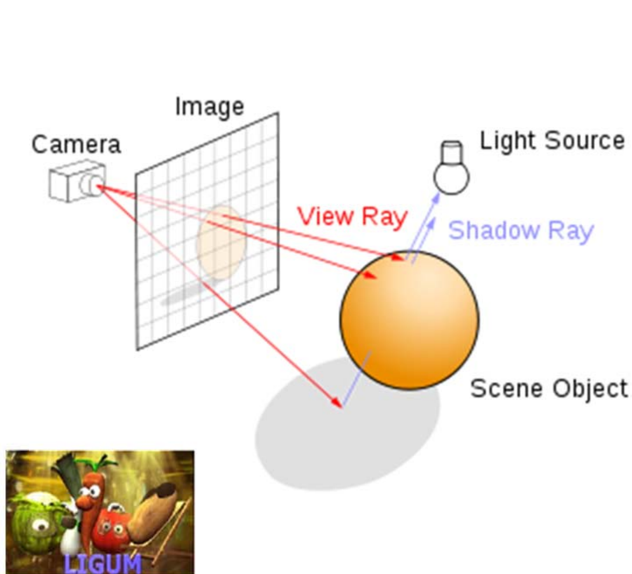
Famille 3: Micropolygon

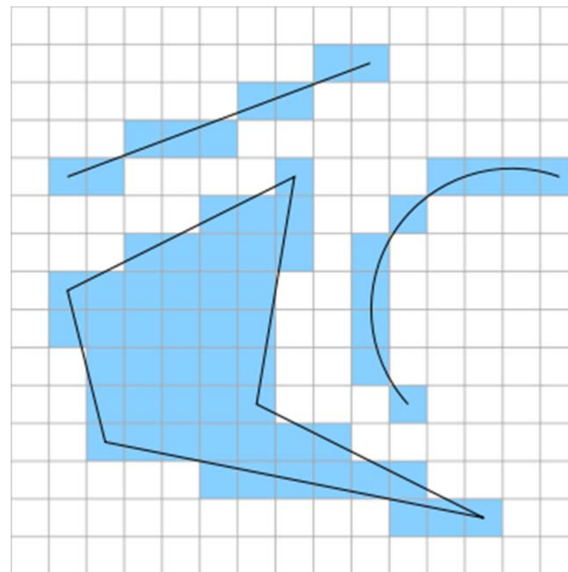
- REYES = algorithme de rastérisation de micropolygon
- Renderman = un standard pour permettre les applications de modélisation / animation de communiquer avec un moteur de rendu
  - N'exige pas un moteur de rendu de micropolygon
- PRMan = premier system qui conformait à la spécification *Renderman*, développer par PIXAR
  - Actuellement un moteur hybrid de micropolygon et lancer de rayons
- REYES est l'algorithme <u>le plus utilisé</u> pour le rendu des films*

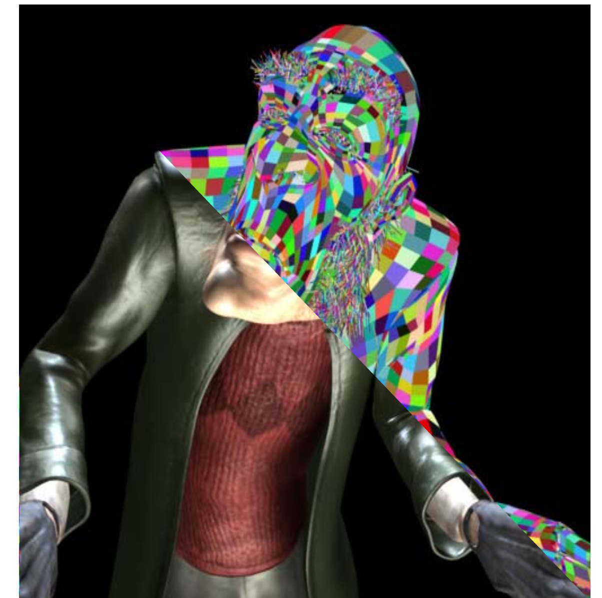
Université de Montréal

Systèmes de rendu: une survol historique

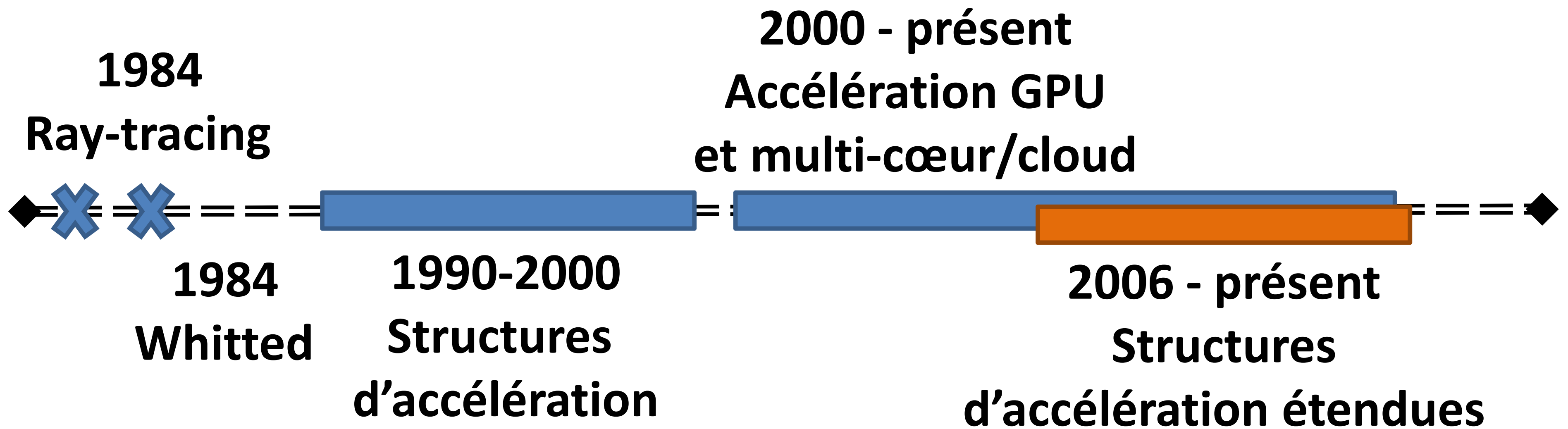


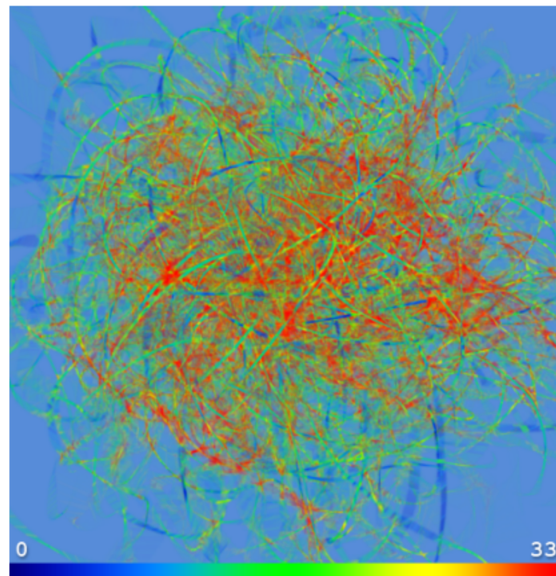


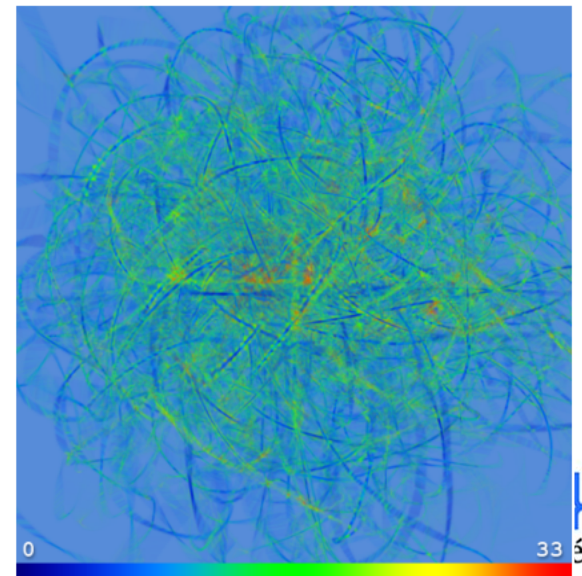




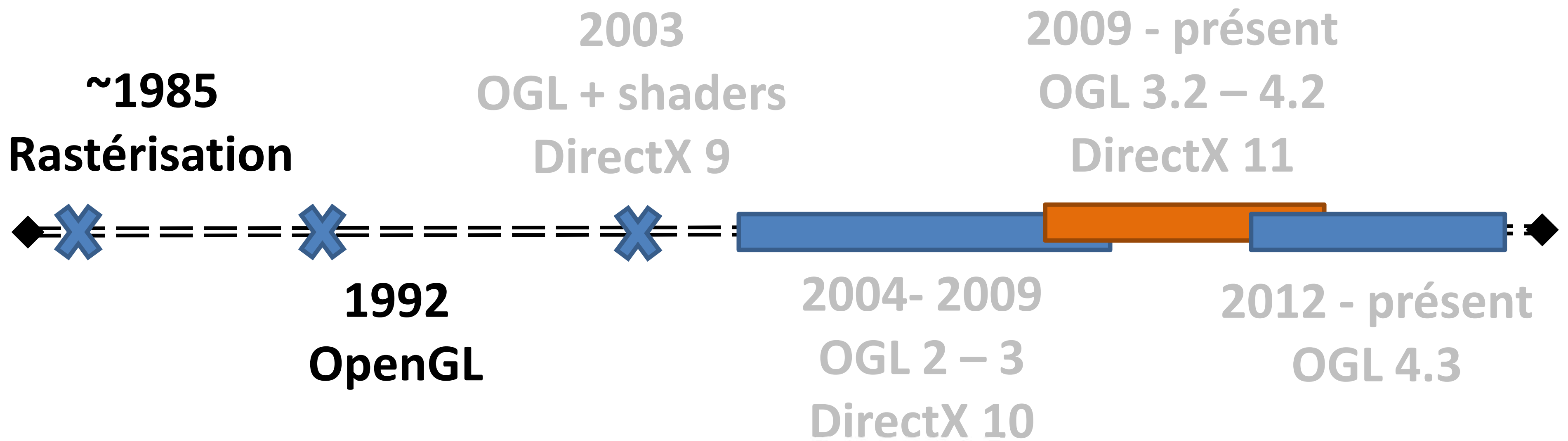
Lancer de rayons: évolution historique

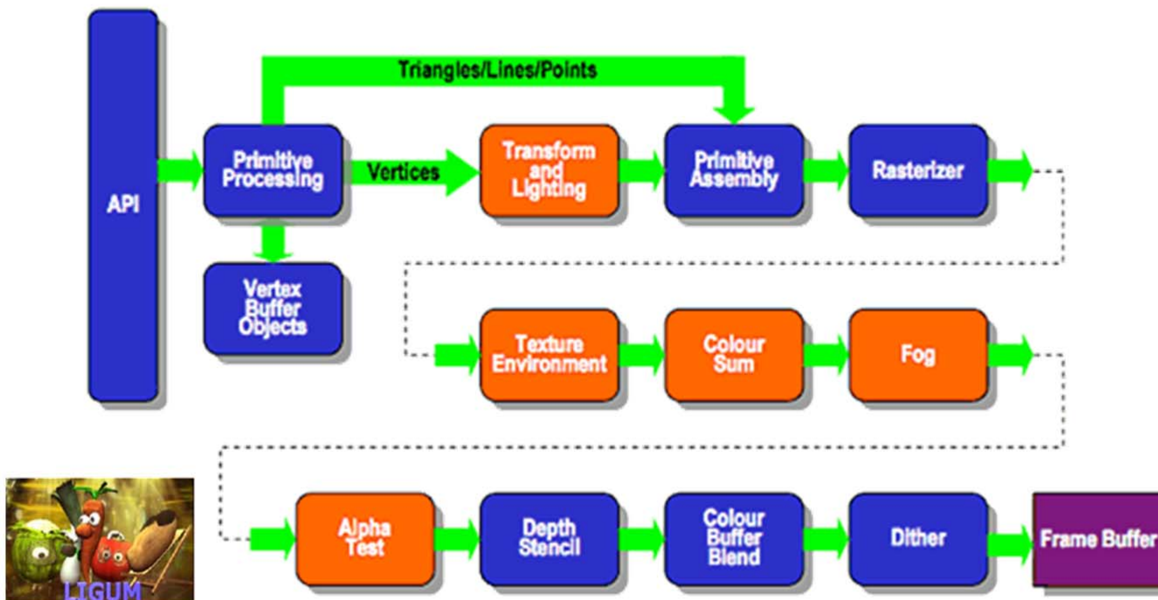


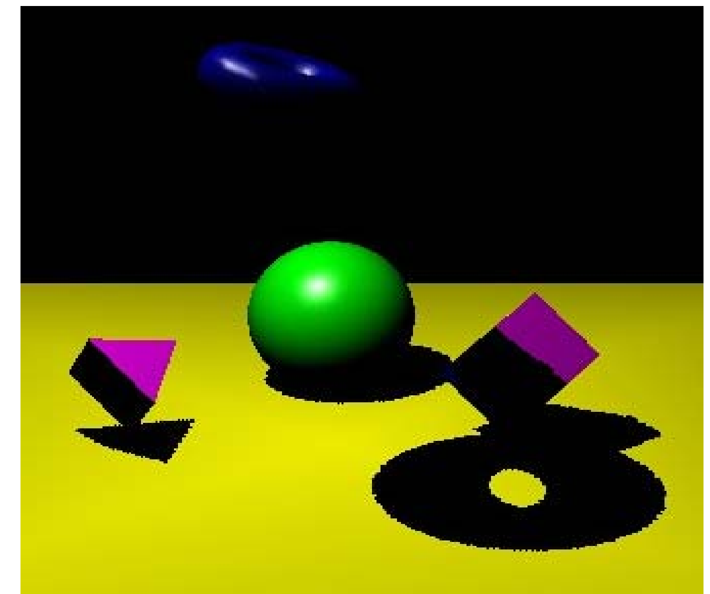




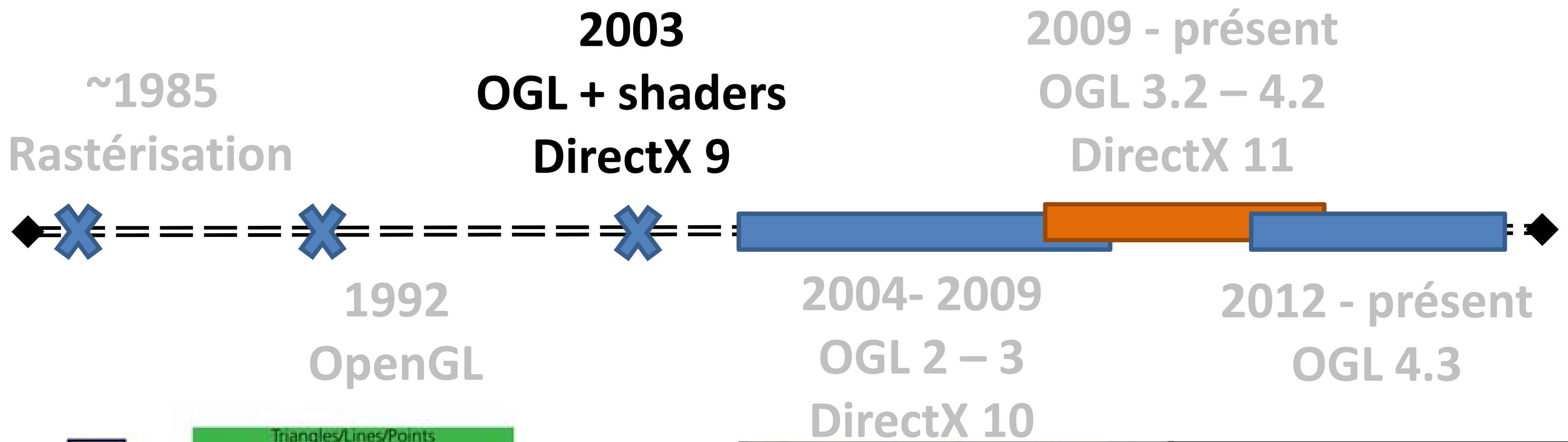
Rastérisation: évolution historique

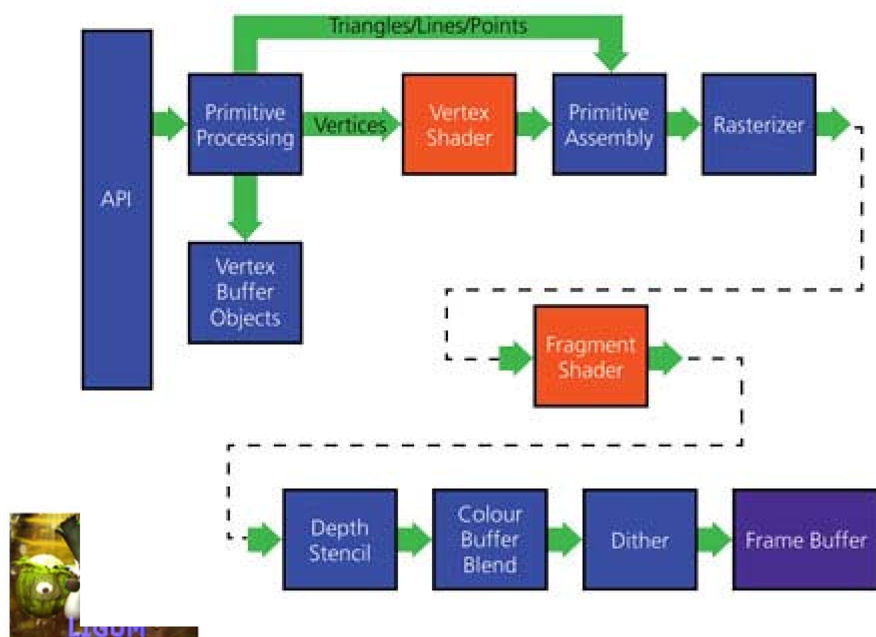


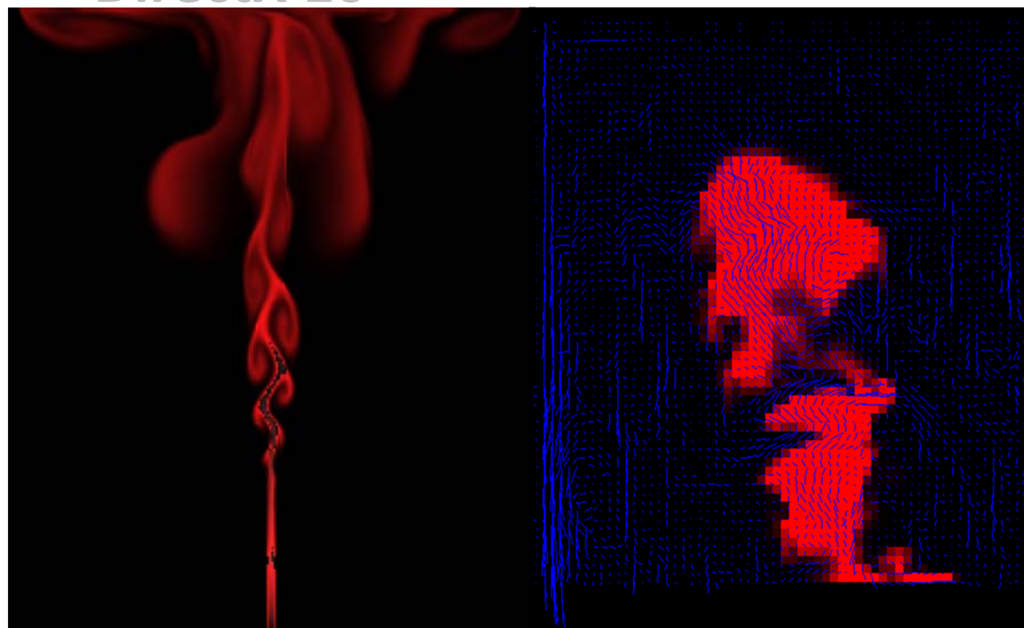




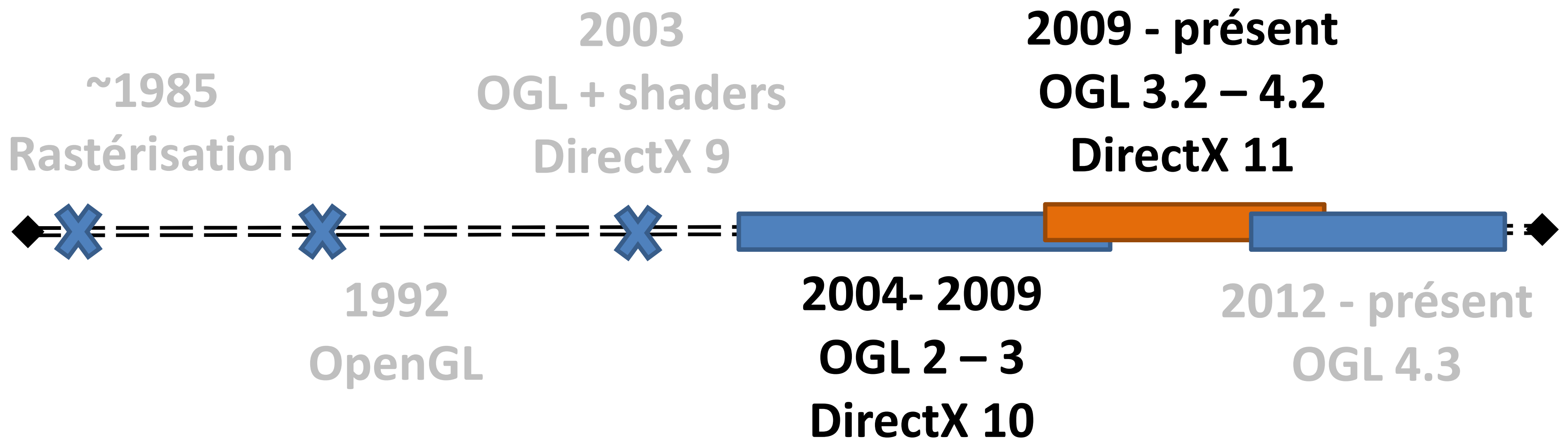
Rastérisation: évolution historique

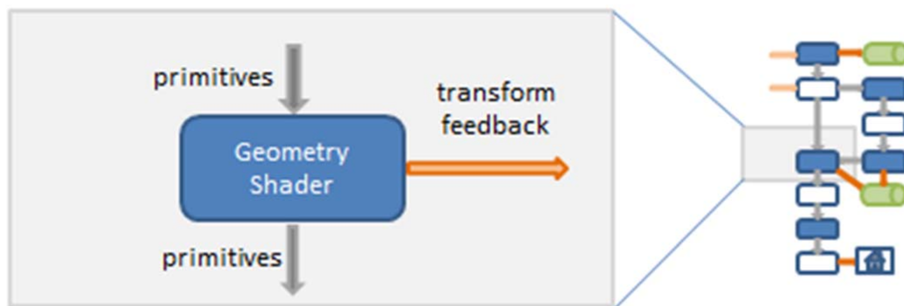


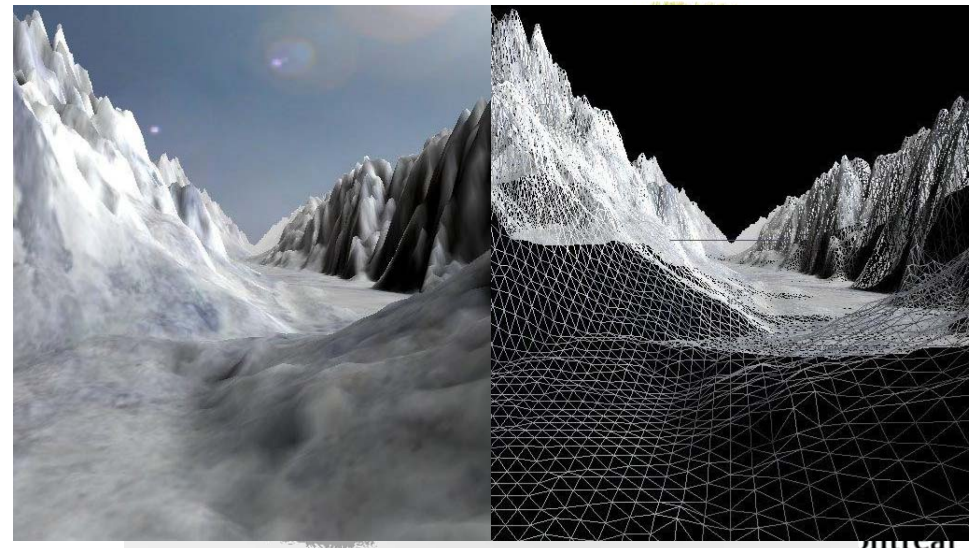




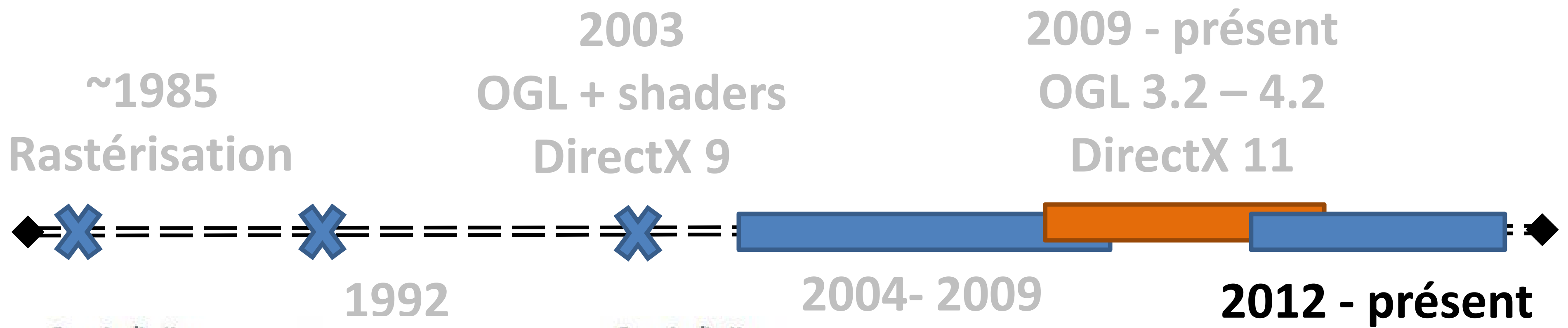
Rastérisation: évolution historique

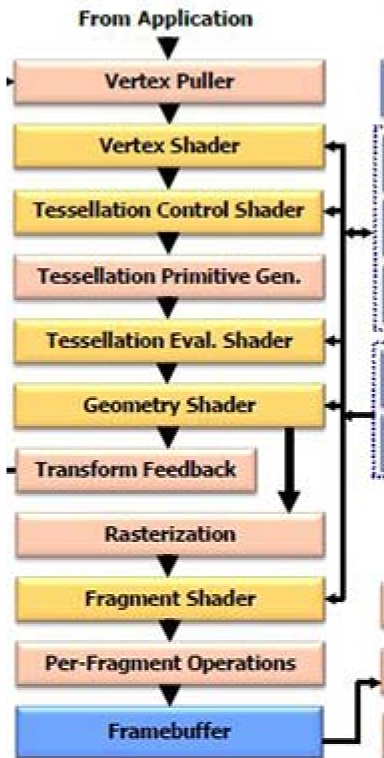


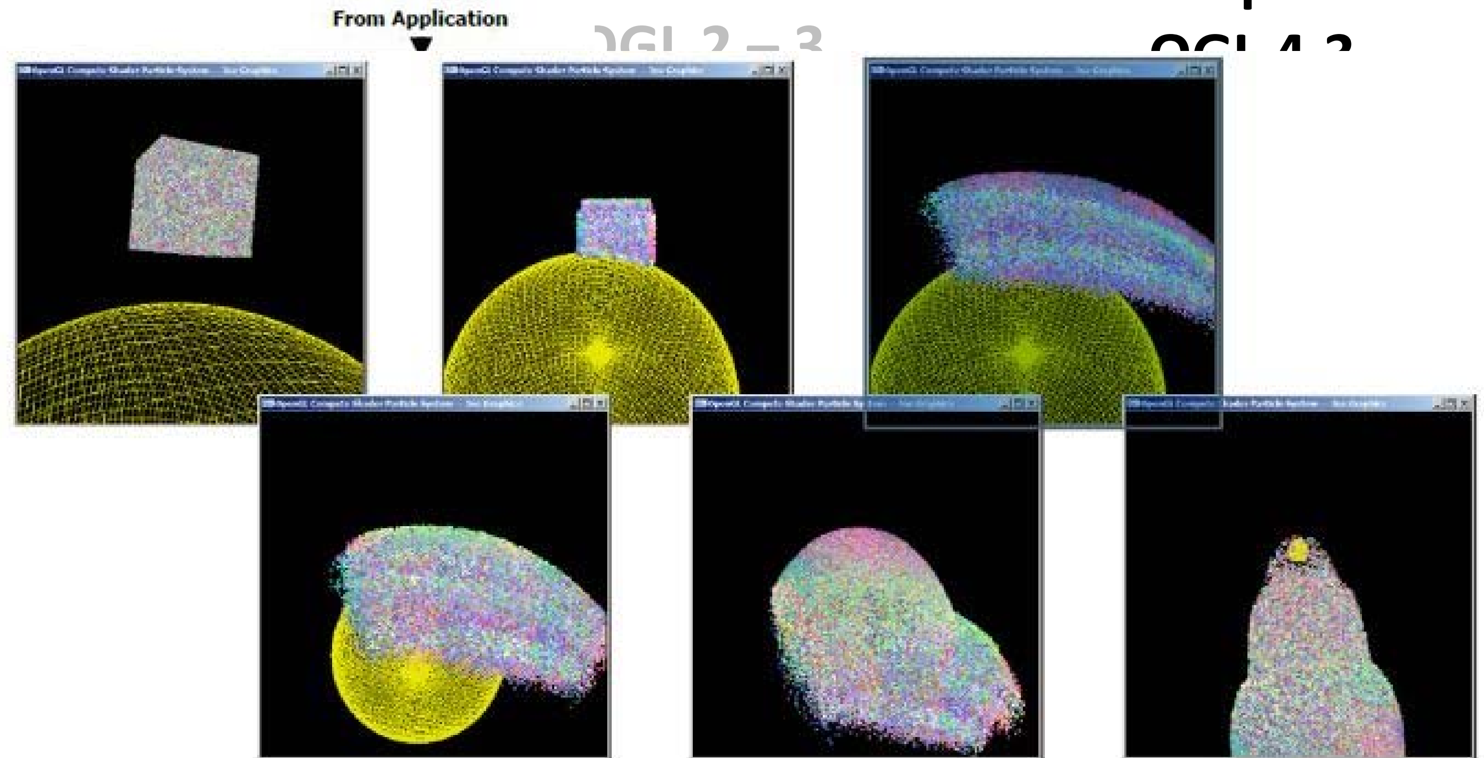




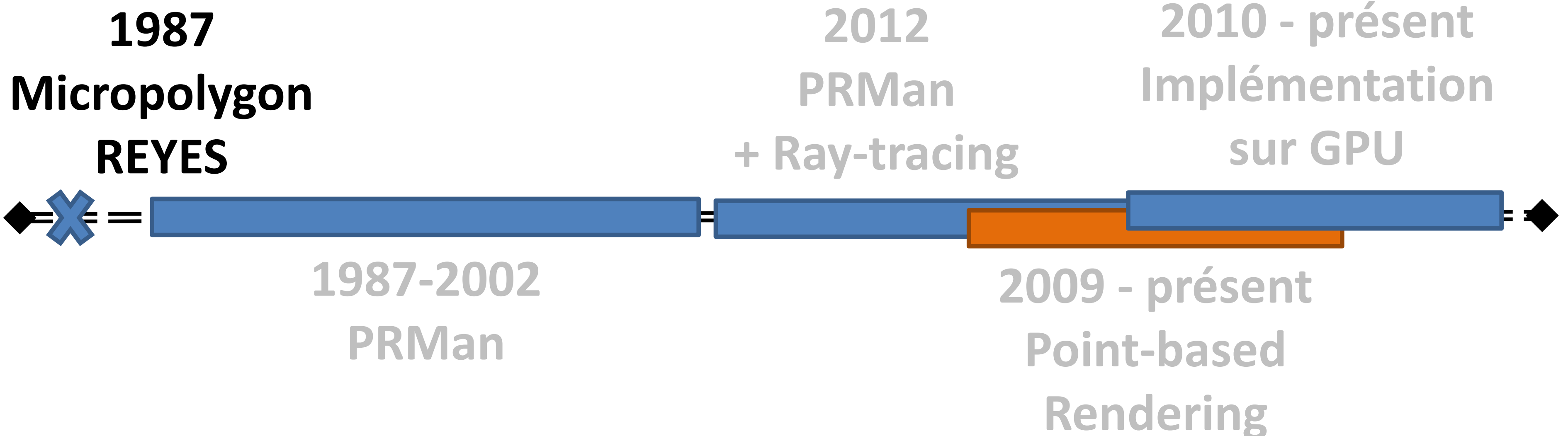
Rastérisation: évolution historique

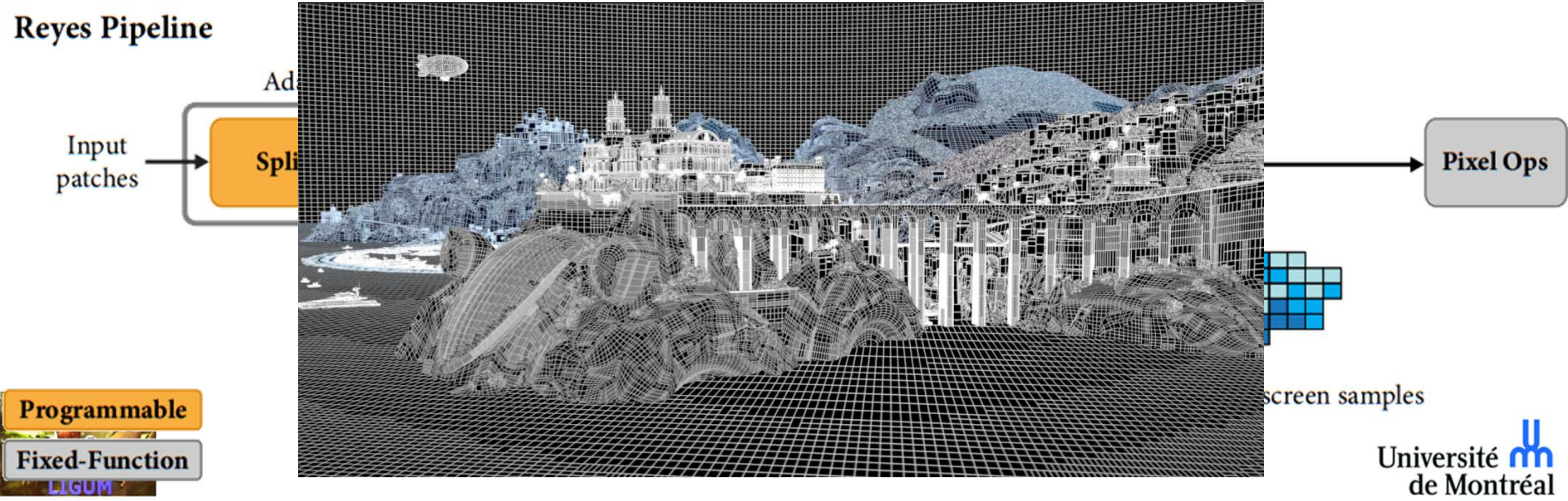




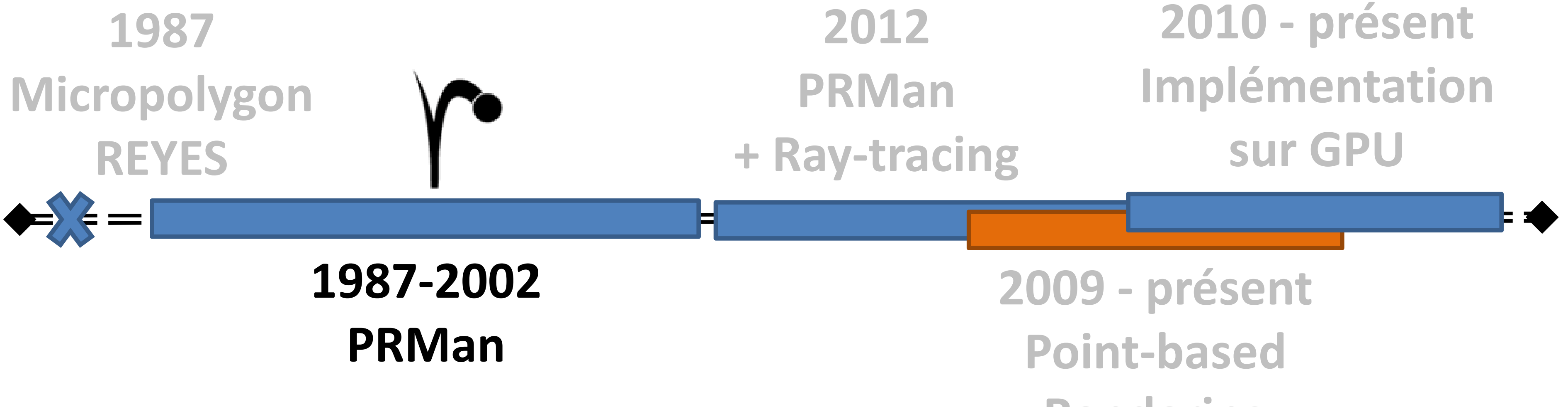


Micropolygon: évolution historique





Micropolygon: évolution historique



23 Years of Pixar's RenderMan

"The original motivation to do the research that led to these technological discoveries was to expand the range of possibilities for storytellers."
Ed Catmull, President of Walt Disney and Pixar Animation Studios

The Commercial Years
From 1990 to 1996 Pixar created many innovative 3D commercials for television, while developing its rendering technology — Photorealistic RenderMan.

Tin Toy · knick knack · Toy Story · Geri's game · a bug's life · Toy Story 2 · for the birds · Monsters, Inc. · Finding Nemo · The Incredibles · Boundin' · One Man Band · Cars · Ratatouille · Lifted

1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007

RI Spec 3.0 Published
With the first use of the word "RenderMan," the RenderMan Interface Specification was the culmination of years of Pixar's rendering development and set the standard for describing 3D data.

RenderMan Toolkit 3.0
Stochastic Sampling
REYES Algorithm
Micropolygons

RenderMan Toolkit 3.2
Pixar "Looks" Support
Zthreshold Shadow Opacity
Better Memory Utilization

RenderMan Toolkit 3.1
RenderMan Interface Bytestream (RIB)
Constructive Solid Geometry
Vector RenderMan

The RenderMan Companion
Publication of the classic RenderMan reference guide, a guide which remains essential for any RenderMan library.

RenderMan Toolkit 3.3
"netrender"

Achievement Award
Scientific & Engineering Achievement Award® from the Academy of Motion Picture Arts and Sciences.

RenderMan Toolkit 3.5
64 Bit Processor Support
Filterstep

RenderMan Toolkit 3.4
Trim Curves
Extreme Displacement
Vertex Motion blur

RenderMan Toolkit 3.7
Procedural Primitives
RiPoints & RiCurves
Level of Detail

RenderMan Toolkit 3.6
RIB Archives
Vertex parameters
True RiSphere primitive

RenderMan Toolkit 3.9
Blobby Implicit Surfaces
Accumulated Opacity Culling
Centered Derivatives

RenderMan Toolkit 3.8
Subdivision Surfaces
Arbitrary Output Variables
DSO Shadeops

RenderMan Wins an Oscar®

RenderMan Pro Server 10.0
Facevarying Class Specifier
Conditional RI Evaluation
Non-Raster Oriented Dicing

RenderMan Pro Server 11.0
Ray Tracing
Ambient Occlusion
Deep Shadow Maps

RenderMan Pro Server 11.5
Scene Analysis Acceleration
Hider Subpixel Output
Deep Shadow API

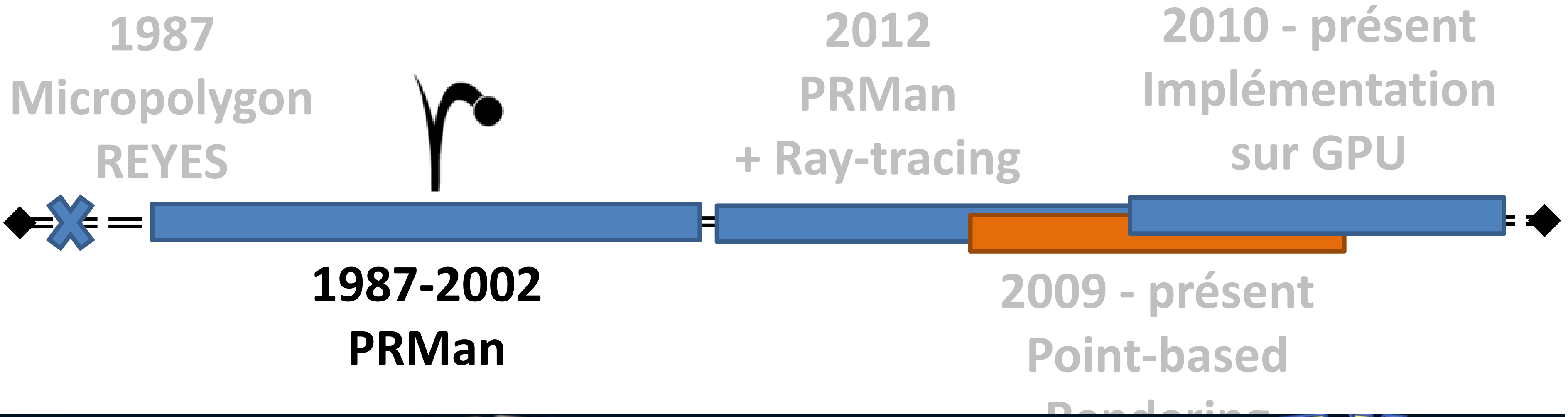
RenderMan Pro Server 12.0
3D BrickMaps
OpenEXR Support
Ri Filters

RenderMan Pro Server 12.5
Accelerated Ray Tracing
Hierarchical Subdivs
Point Cloud API

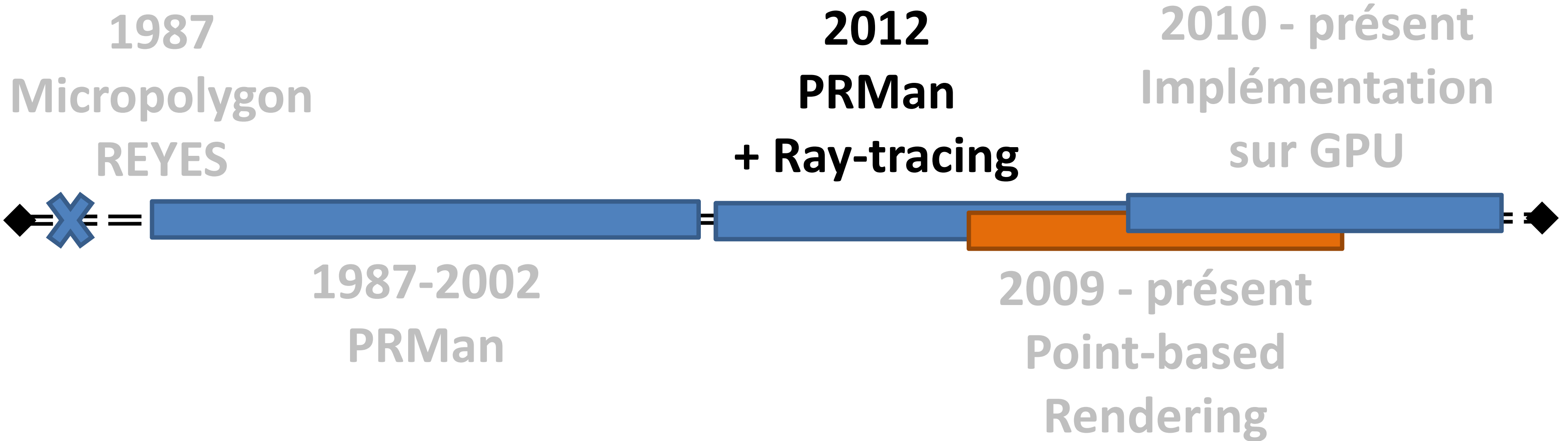
RenderMan Pro Server 13.0
Multi-Threaded Rendering
Brick Maps as Geometry
Point Based Colorbleeding

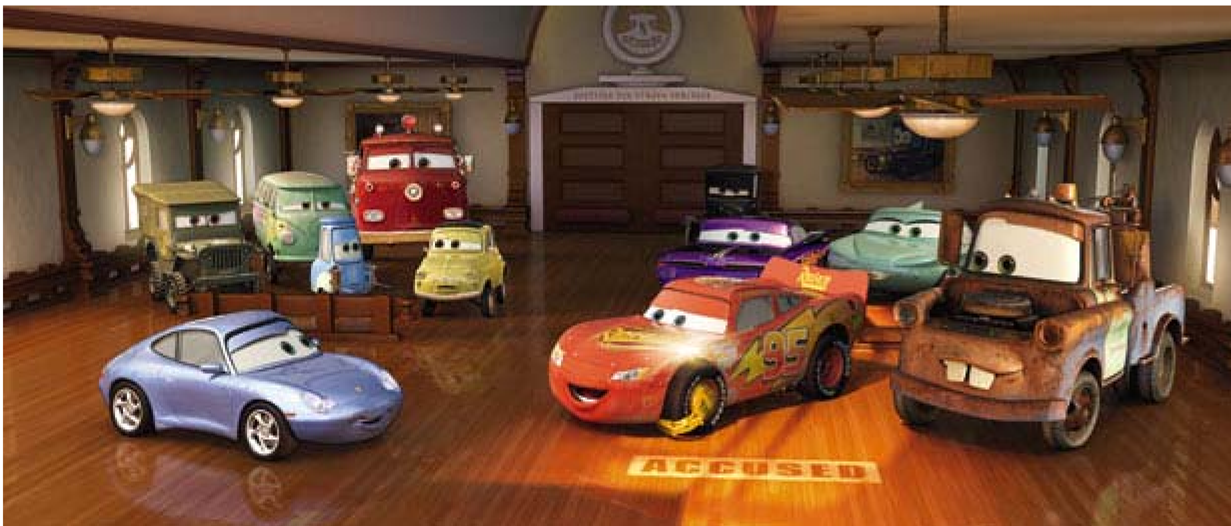
RenderMan Pro S
Stereo Multi-C
Shader Objects
Multi-Threaded

Micropolygon: évolution historique

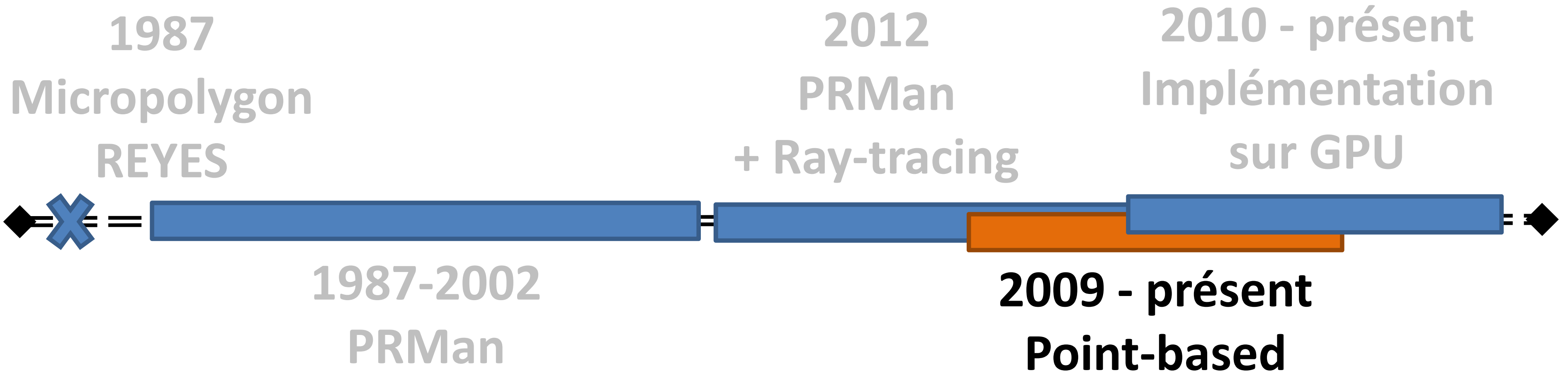


Micropolygon: évolution historique

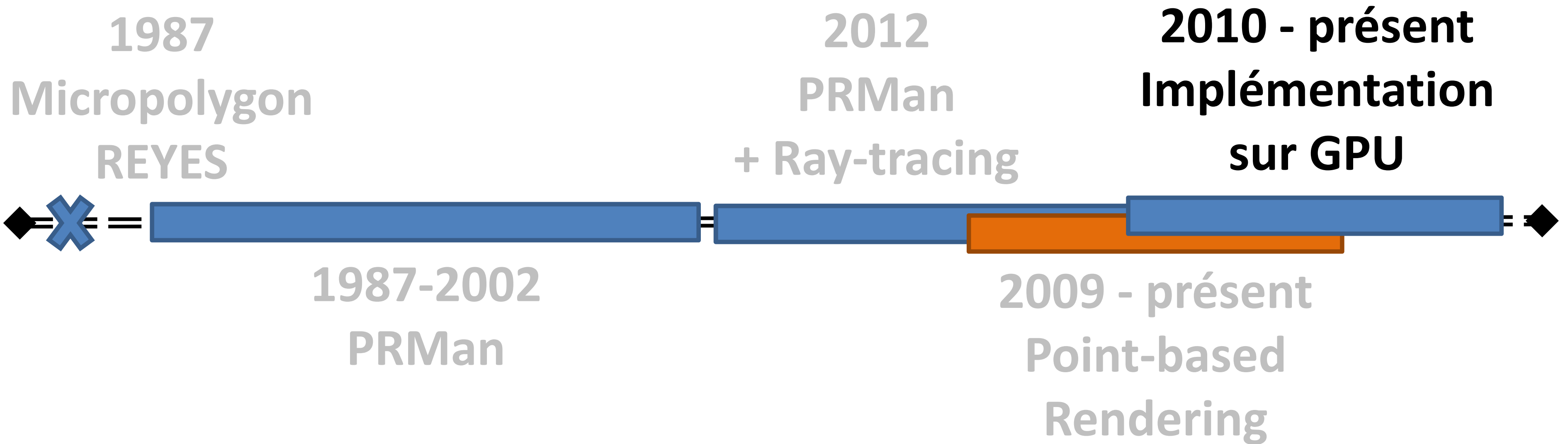


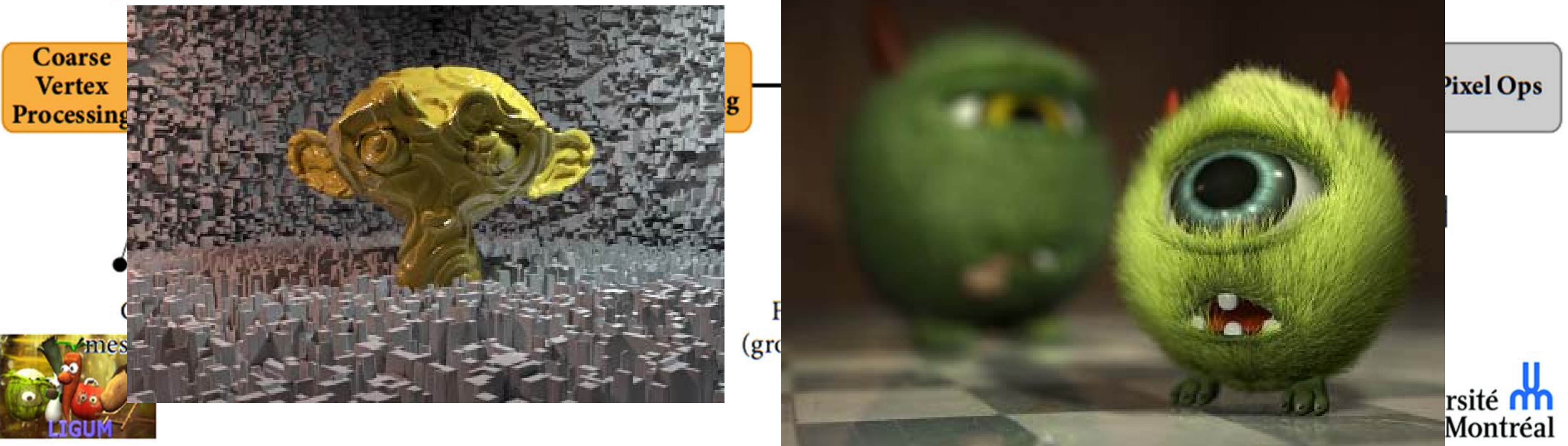


Micropolygon: évolution historique



Micropolygon: évolution historique





Débat religieux: croyez-vous en Rayons ou en Rastériseur?

- Question piège: ça dépend sur le bût de ton système, les contraintes, les types d'effets, etc.

http://c0de517e.blogspot.ca/2011/09/raytracing-myths.html

https://plus.google.com/105941772928736706414/posts/TFm2kjThKSF

http://www.dailytech.com/John+Carmack+on+Ray+Tracing+vs+Rasterization/article11095.htm

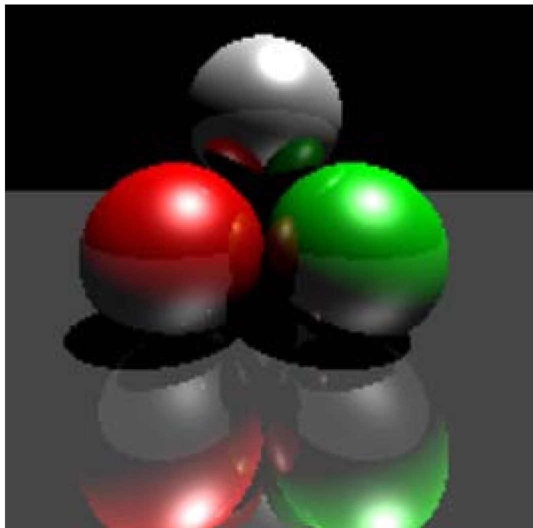
http://www.scarydevil.com/~peter/io/raytracing-vs-rasterization.html

- Quels sont les tendances actuels en industrie?
- … et en recherche académique?

Université de Montréal

Évolution et extensions

- Chaque architecture a évolué de leur implémentation de base au fil de temps afin de supporter de plus en plus de fonctionnalité



Évolution et extensions

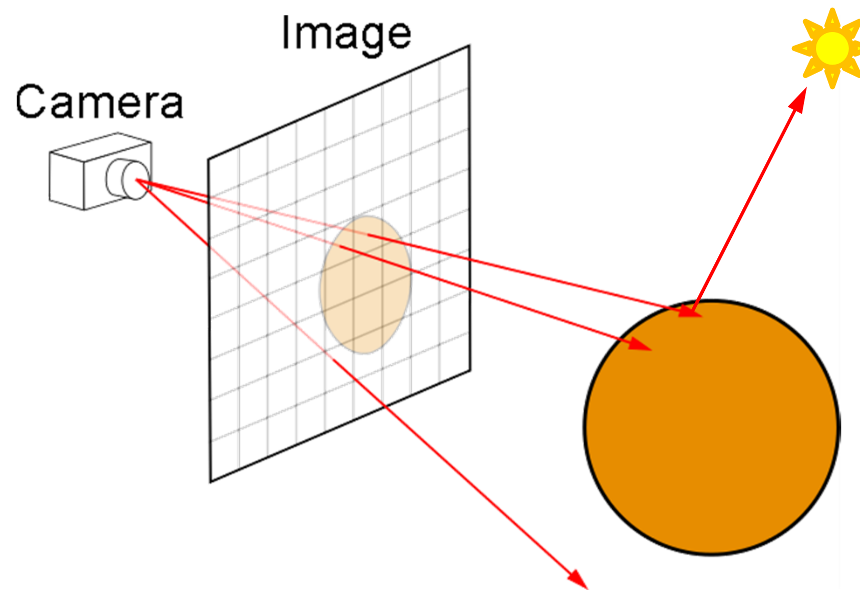
- Chaque architecture a évolué de leur implémentation de base au fil de temps afin de supporter de plus en plus de fonctionnalité

Évolution et extensions

- Chaque architecture a évolué de leur implémentation de base au fil de temps afin de supporter de plus en plus de fonctionnalité

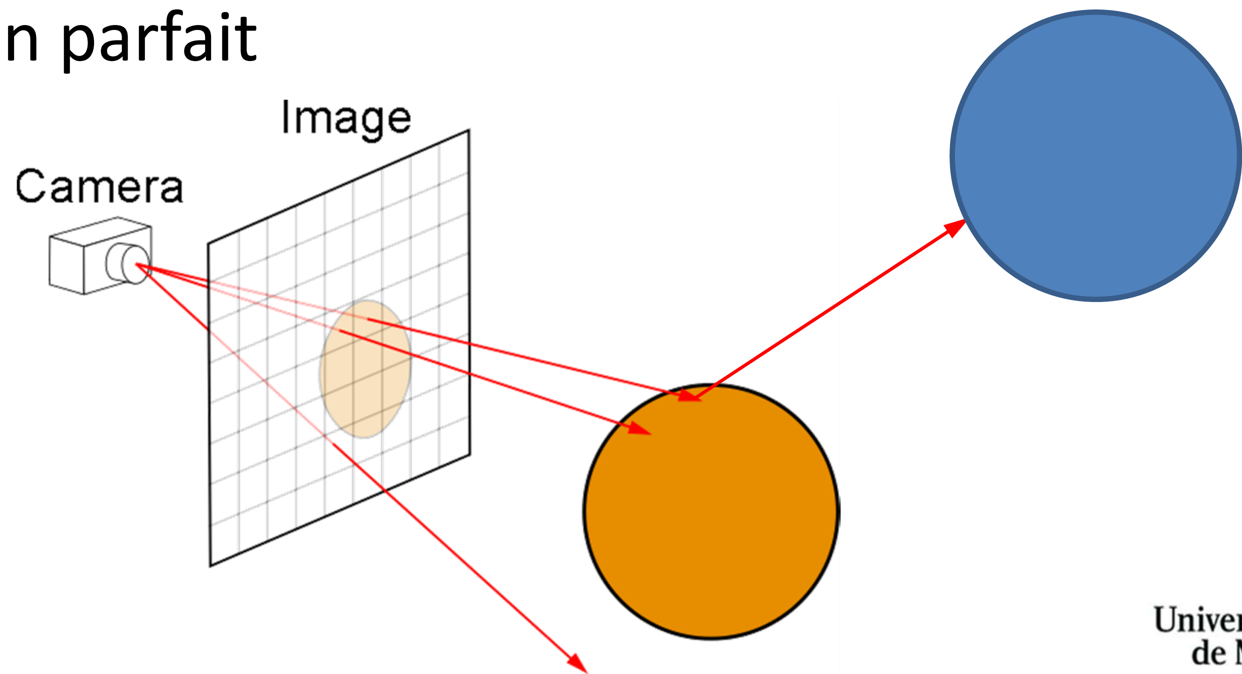
Évolution & extensions: lancer de rayons

- Un des premier extensions dans l'évolution des lanceurs de rayons est le système de Whitted
  - Fournit une seule couche de recursion, permettant la simulation des ombres durs et des réflexion / réfraction parfait



Évolution & extensions: lancer de rayons

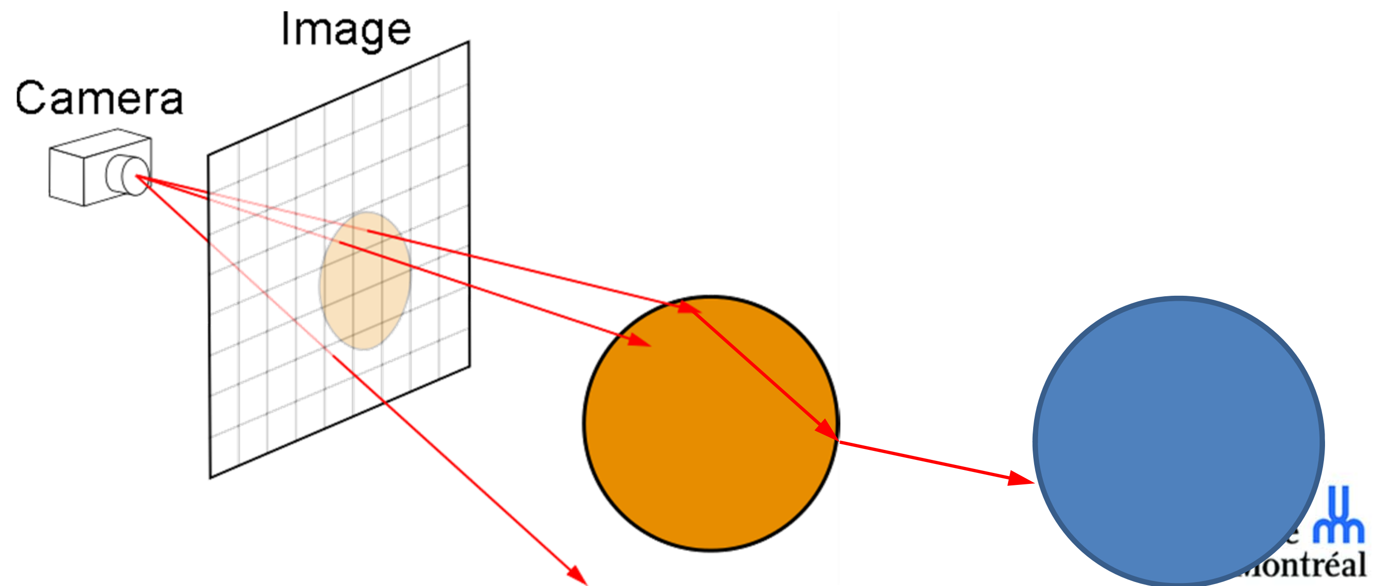
- Un des premier extensions dans l'évolution des lanceurs de rayons est le système de Whitted
  - Fournit une seule couche de recursion, permettant la simulation des ombres durs et des réflexion / réfraction parfait



Université de Montréal

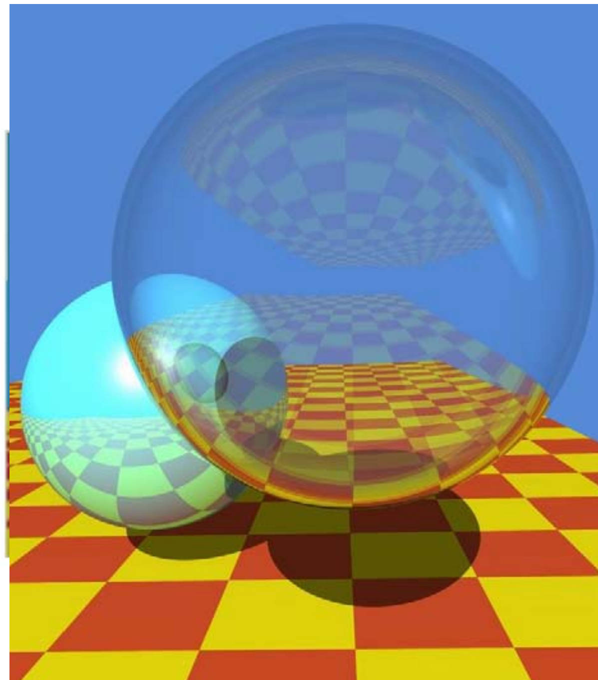
Évolution & extensions: lancer de rayons

- Un des premier extensions dans l'évolution des lanceurs de rayons est le système de Whitted
  - Fournit une seule couche de recursion, permettant la simulation des ombres durs et des réflexion / réfraction parfait



Évolution & extensions: lancer de rayons

- Un des premier extensions dans l’évolution des lanceurs de rayons est le système de Whitted
  - Fournit une seule couche de recursion, permettant la simulation des ombres durs et des réflexion / réfraction parfait



Évolution & extensions: lancer de rayons

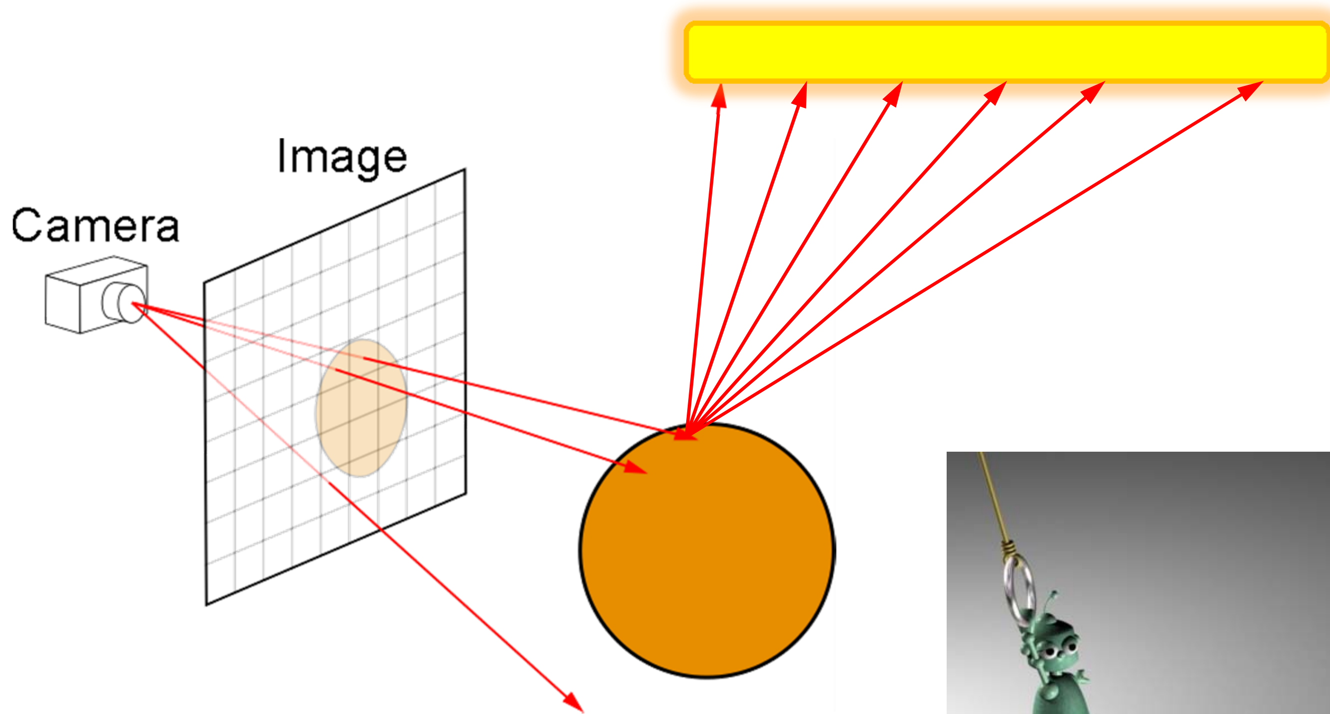
- Il a fallu plusieurs années avant que les gens ont non-seulement formulé l'équation de rendu mais aussi noter qu'un lanceur de rayons pouvais servir à le résoudre*
  – (sous forme d'un estimateur Monte Carlo)

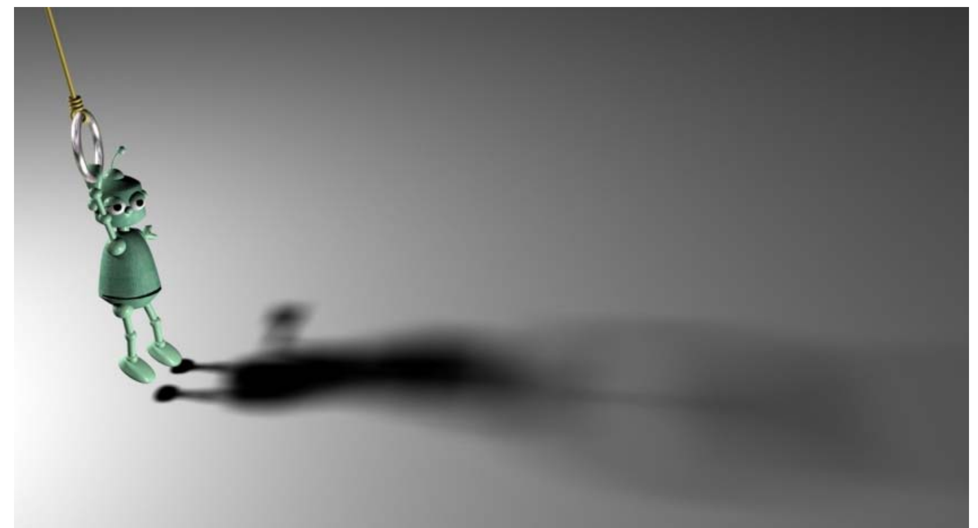
- Plusieurs effets d'illumination avancés peuvent être **facilement** implémentés avec un arbre de rayons récursif et/ou distribué …

(et alors dans un lanceur de rayons)

Évolution & extensions: lancer de rayons

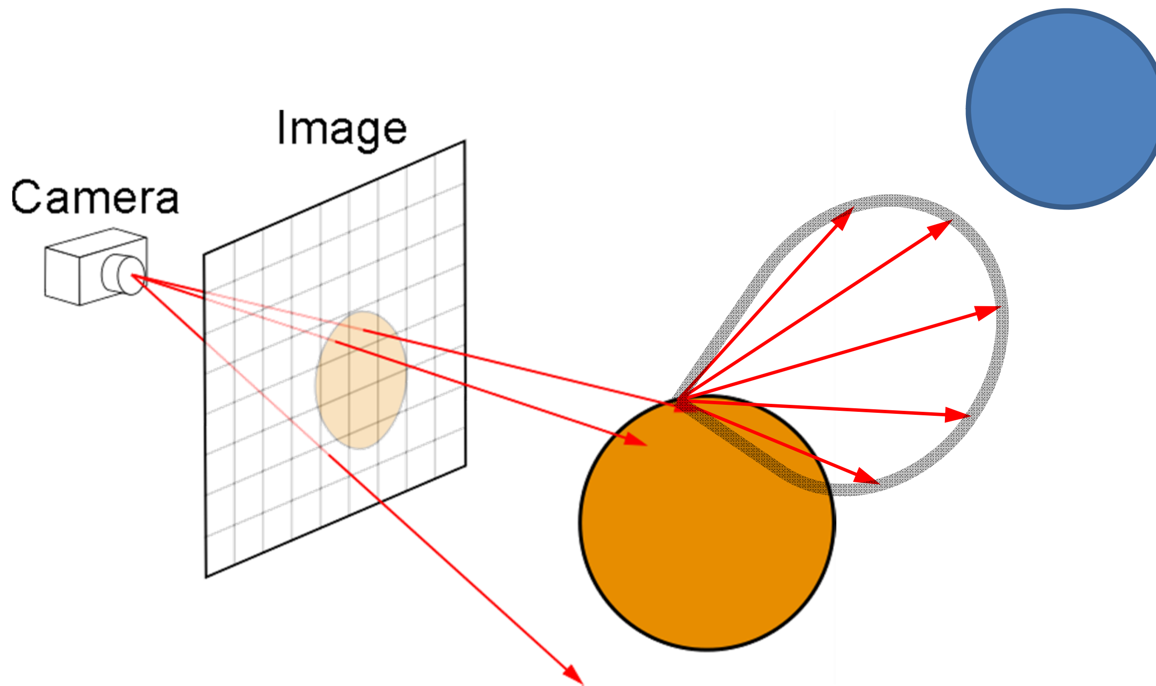
- Ombres étendues

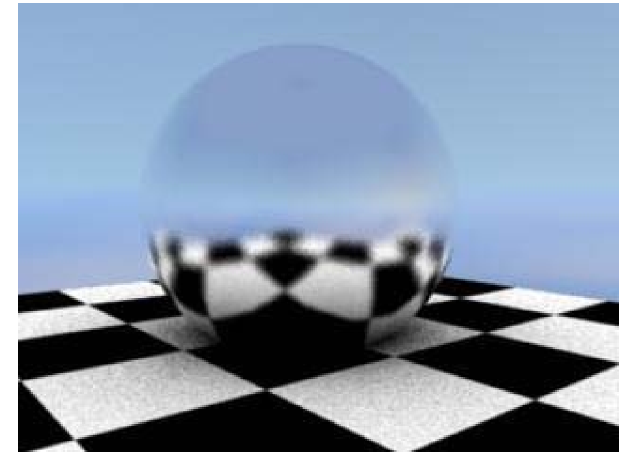


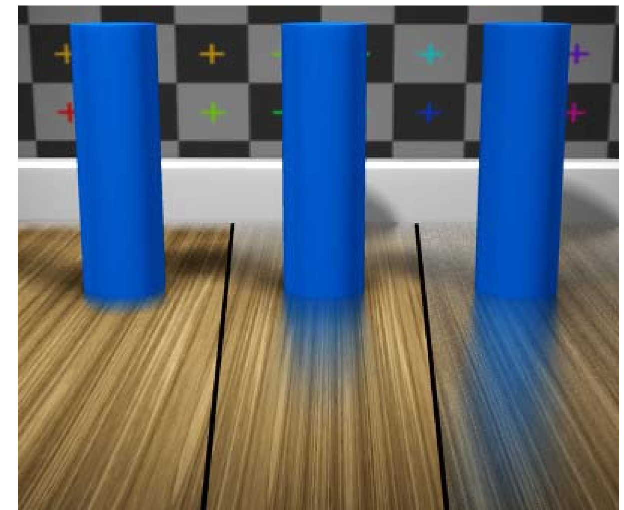


Évolution & extensions: lancer de rayons

- Réflexion *glossy* (distribué)

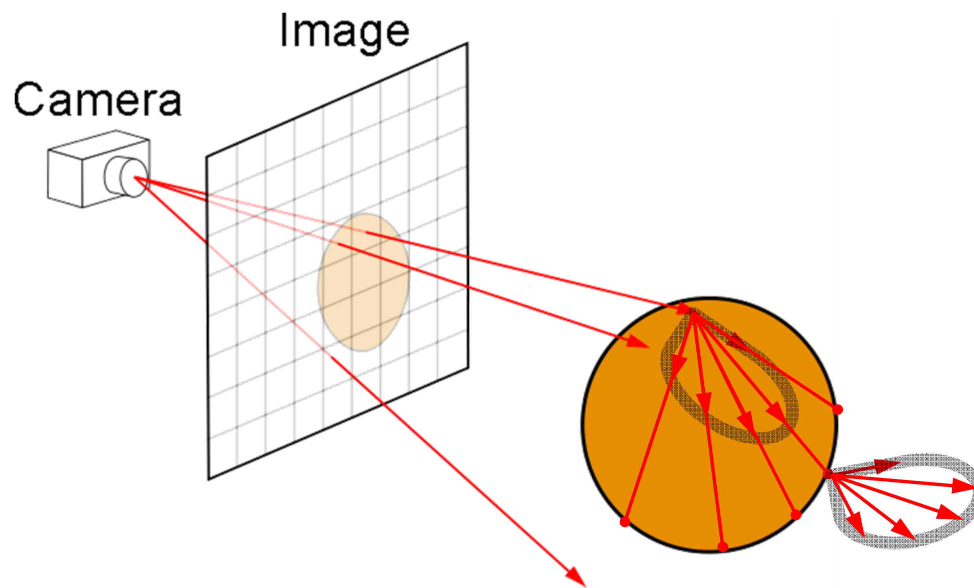


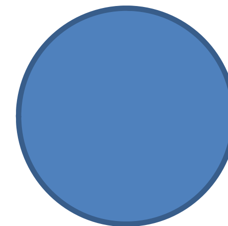




Évolution & extensions: lancer de rayons

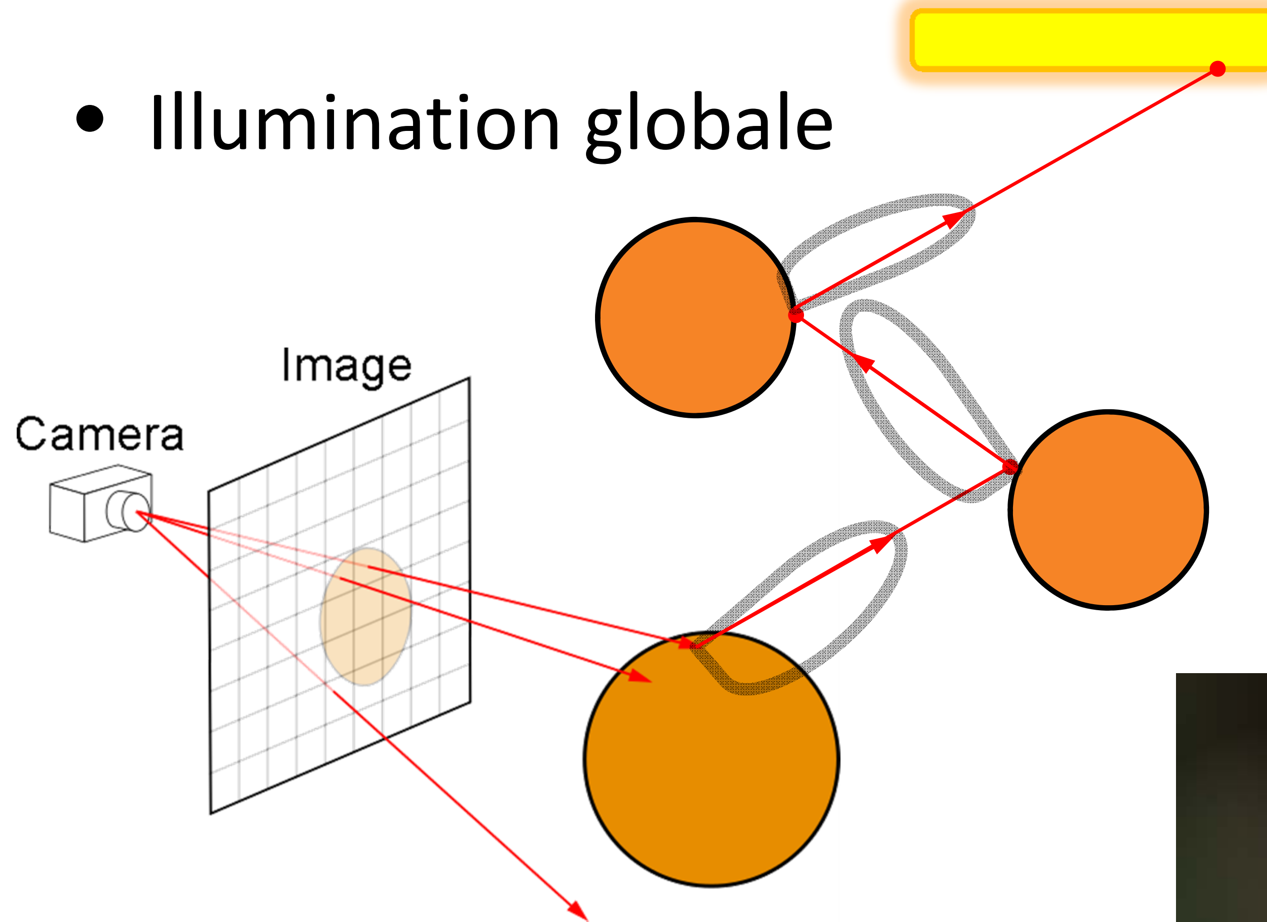
- Transmission/réfraction *glossy* (distribué)





Évolution & extensions: lancer de rayons

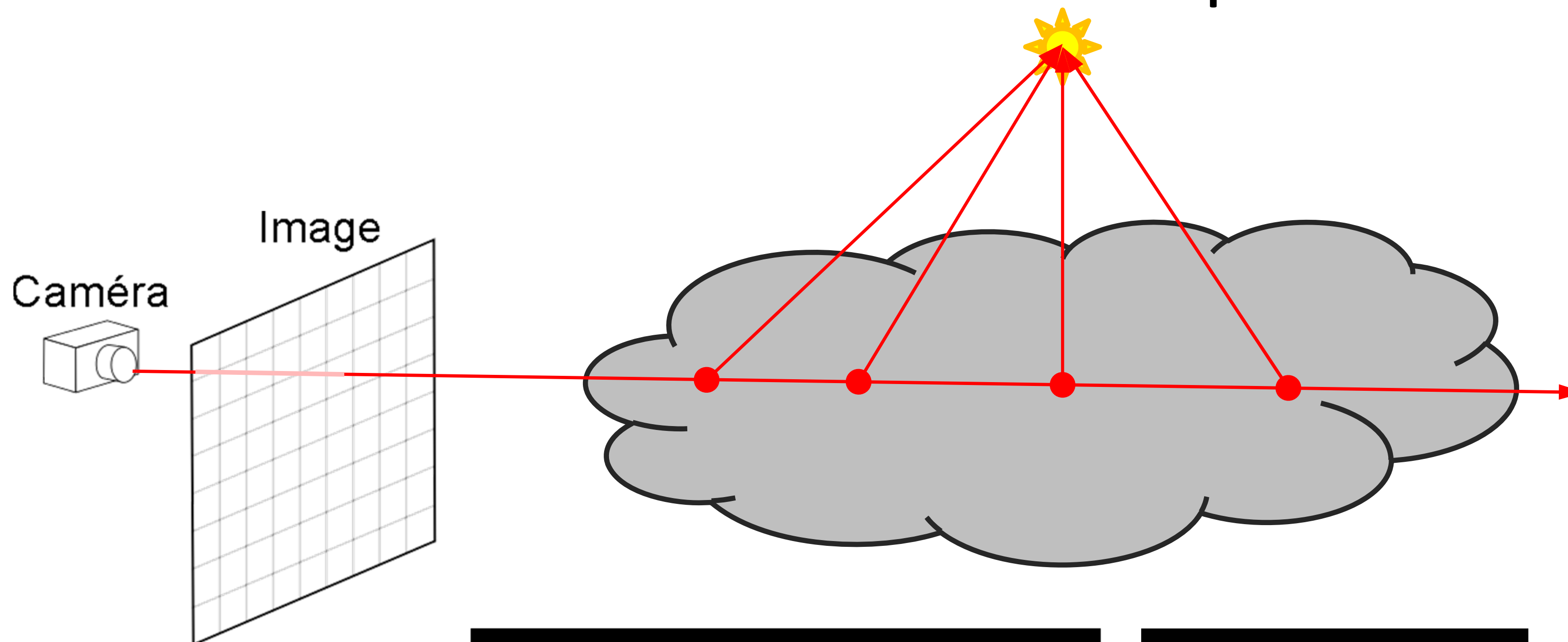
- Illumination globale



de Montreal

Évolution & extensions: lancer de rayons

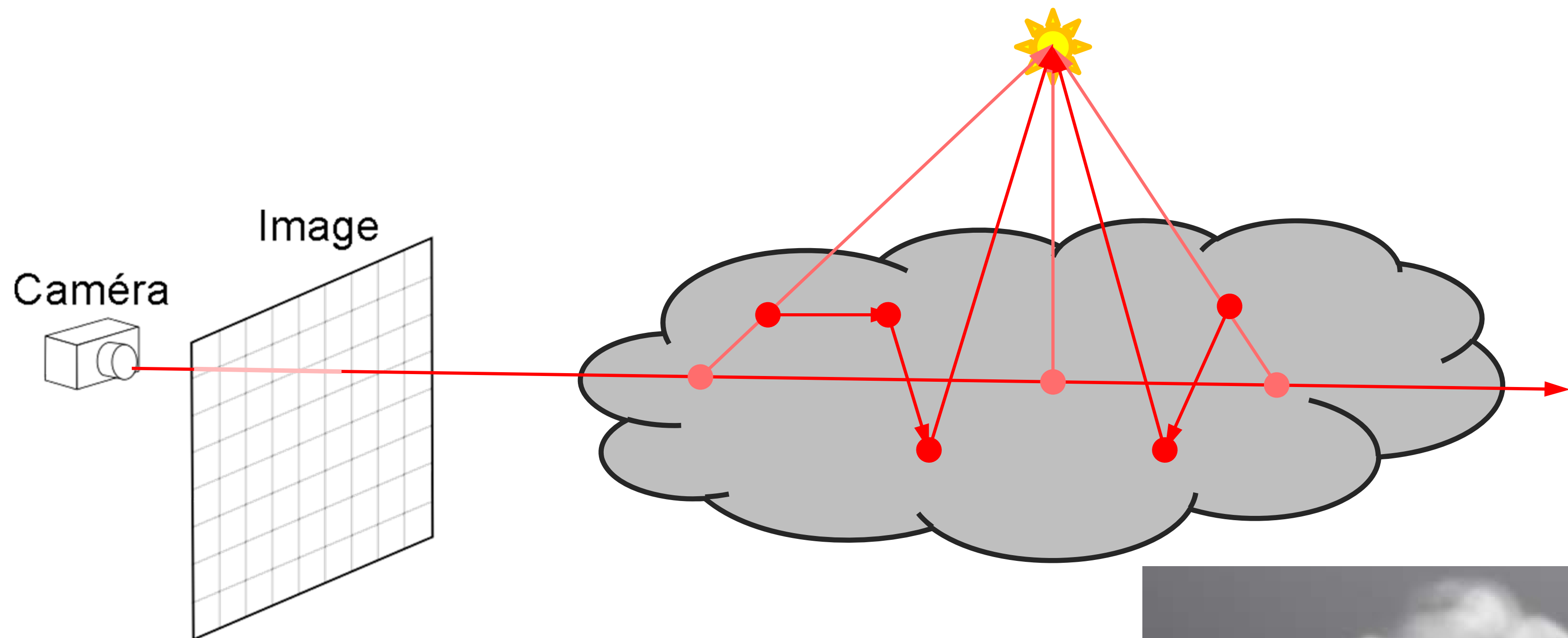
- Illumination directe volumétrique

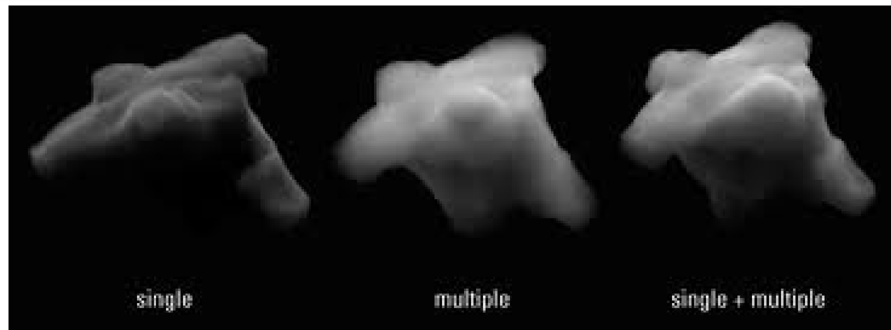


Université de Montréal

Évolution & extensions: lancer de rayons

- Illumination globale volumétrique





Le future: lancer de rayons

- Comme l'incorporation l'illumination complexes est plus simple dans ce système, un des défis fondamentaux se concerne avec les calculs d'intersection:
  - Quand la scène change dynamiquement
  - Quand la géométrique s'approche une résolution sous-pixel
  - Quand la cohérence en espace rayons n'est maintenu
  - Les implémentations sur HW
- Celui-ci comprend la conception des nouveaux structures d'accélération. Par exemple, …
- Les améliorations aux modèles mathématiques d'illumination sont souvent facile à intégrer dans un lanceur de rayons

Université de Montréal

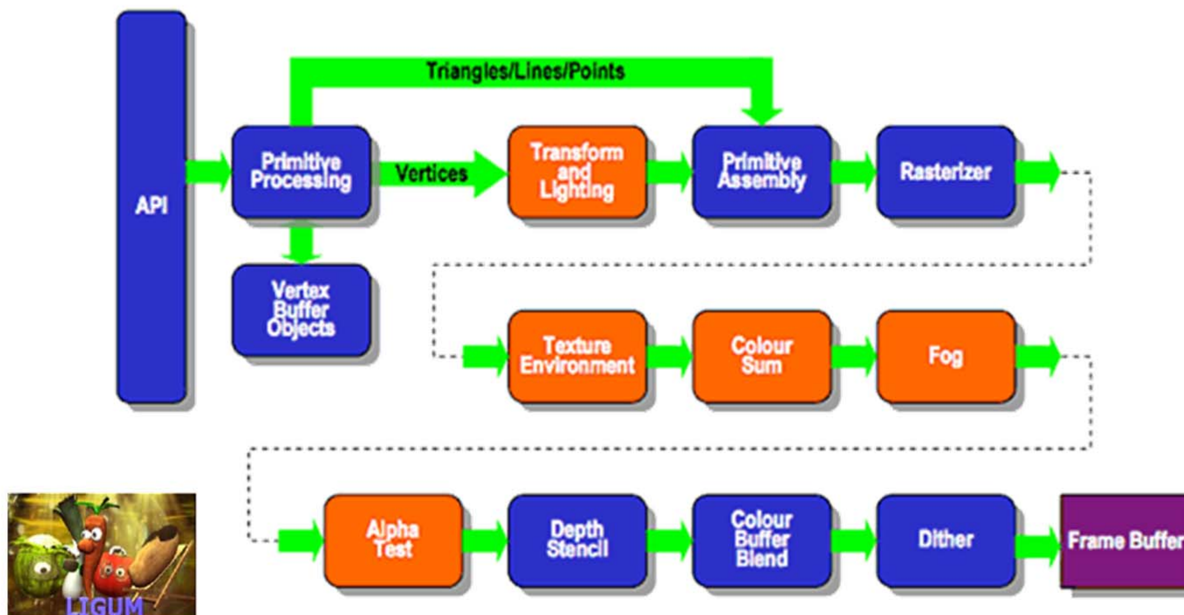
Évolution & extensions: rastérisation

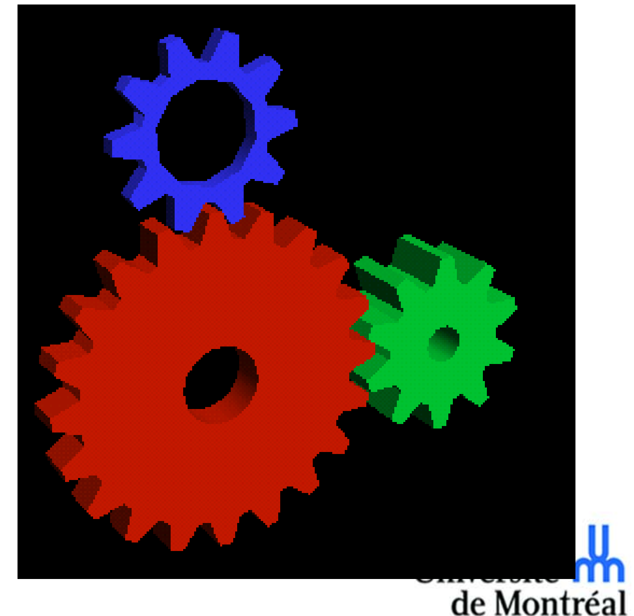
- Un système de rastérisation de base doit convertir entre une représentation vectorielle de la scène à un représentation *raster*/image en:
  - Appliquant des transformations pour apporter tous les objets 3D en espace monde
  - Suivi par un transformation de projection pour les projeter sur le plan de vue
  - L'application des algorithmes de clipping (aussi faisable en espace non-projeter)
  - Et finalement le remplissage et les tests de visibilité

Université de Montréal

Évolution & extensions: rastérisation

- Le premier système de rastérisation standardiser était celle de OpenGL v1 introduit par SGI en 1992
- Le spécification pouvait être réalisé en logiciel ou sur HW selon les étapes **fixed** suivants:





LIGUM

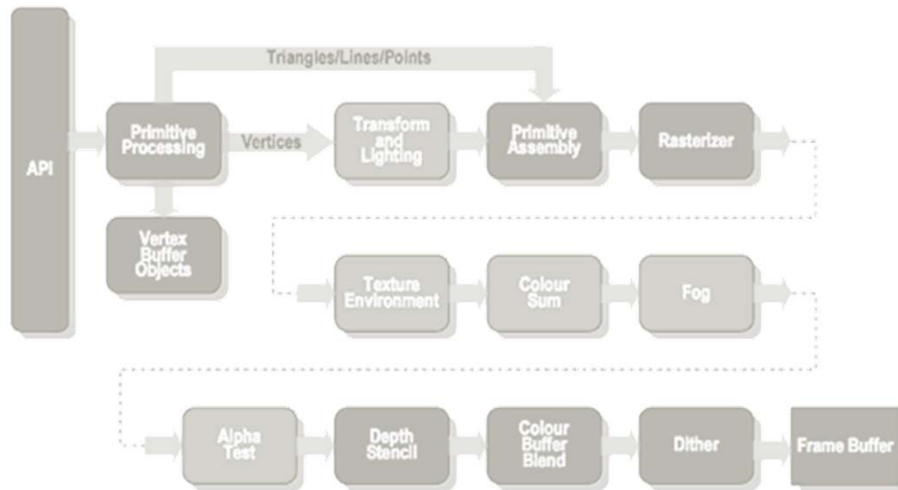
Université de Montréal

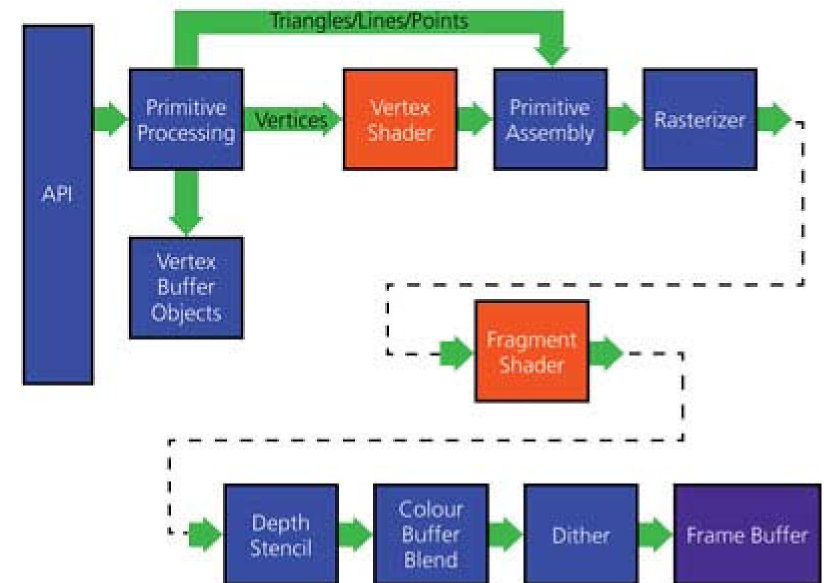
Évolution & extensions: rastérisation

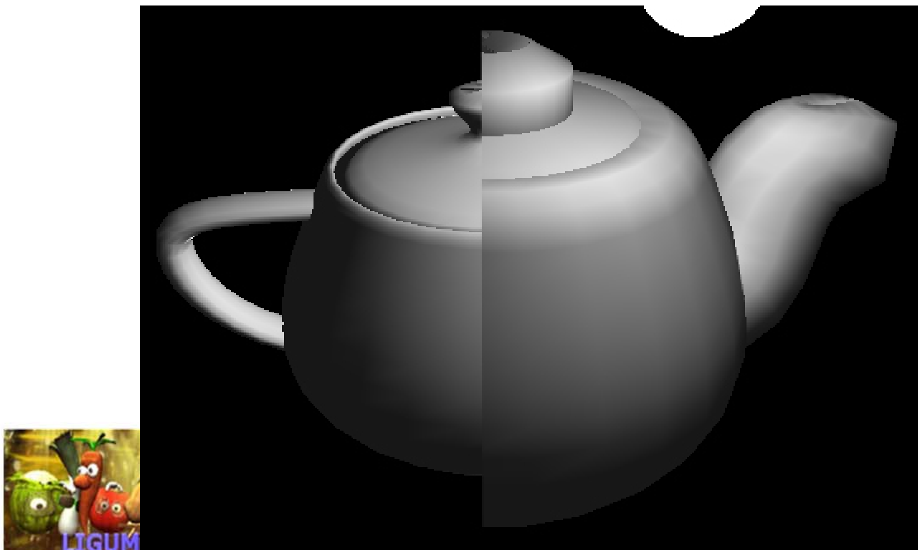
- Le succès des composants programmable dans le pipeline REYES a promu l'utilisation des *texture combiners* pour le calcul des effet faiblement programmable, et cela servira comme catalyseur pour l'évolution du pipeline OGL
- OpenGL v1.5 introduit les premières composantes programmable dans son pipeline:
  - Les vertex shaders et les pixel (fragment) shaders
    - Ces deux composantes peuvent facilement être parallélisé

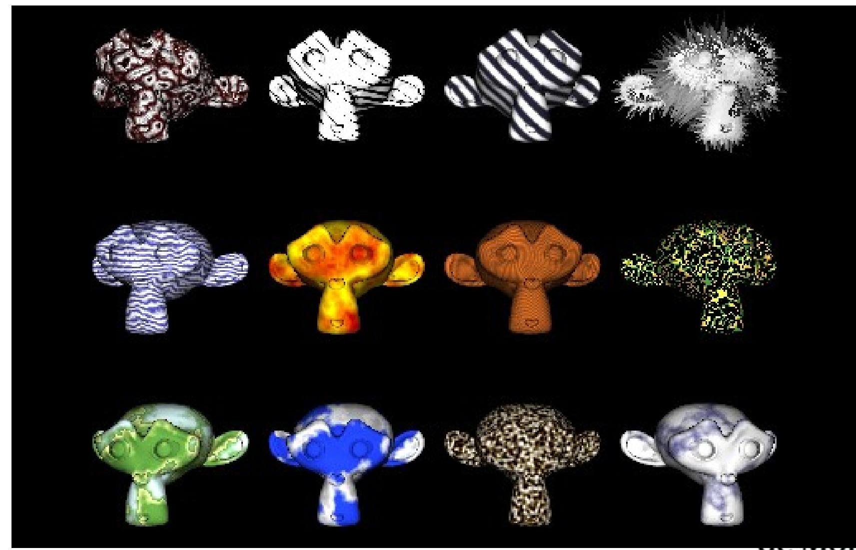
Université de Montréal

Évolution & extensions: rastérisation





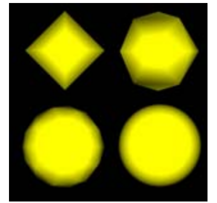


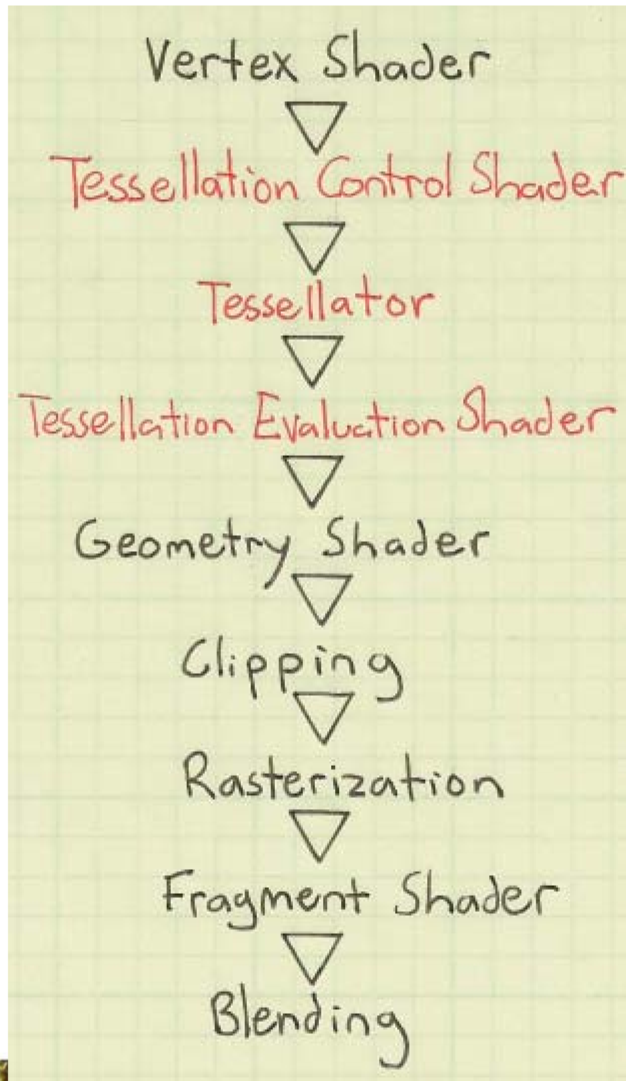


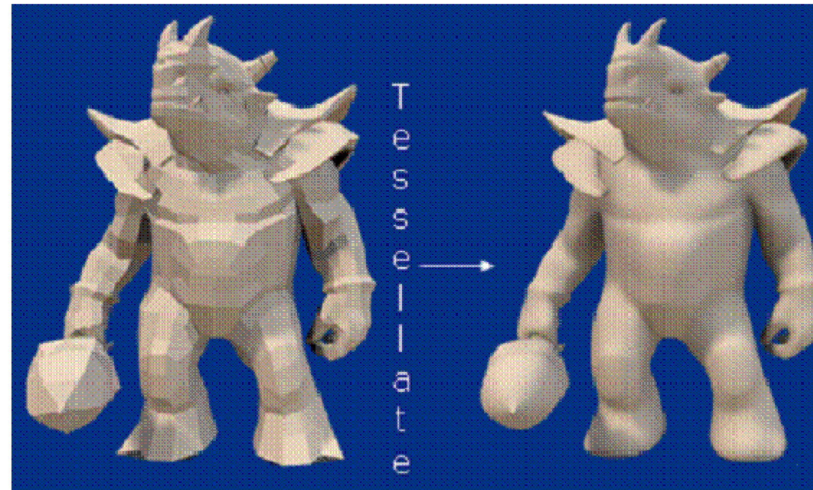
Évolution & extensions: rastérisation

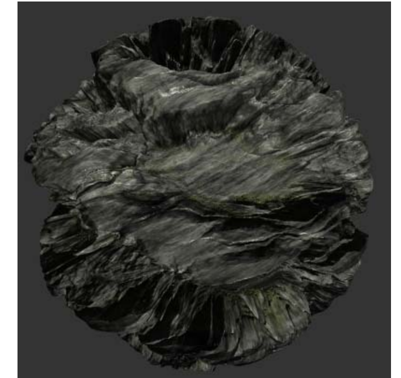
- Les prochaines quelques phases d’évolution du pipeline (programmable) de rastérisation ce concerne avec la génération et modification procédurale de géométrie dans la scène
  - Les choses commencent à ressemble à un moteur de micropolygon…
- Les versions OGL 3.2 et 4.0 ont introduit les *geometry shaders*, et les *tessellation-control* et *tessellation-evaluation shaders**

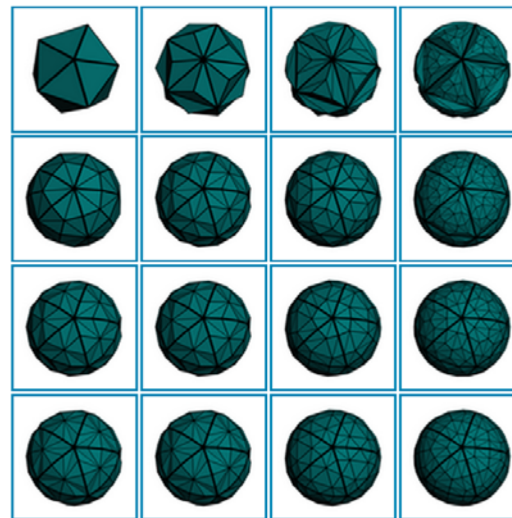
Évolution & extensions: rastérisation











Université de Montréal

Évolution & extensions: rastérisation

- Un point un peu dehors le sujet est le **GPGPU**
- Les développeurs d'infographie, étant un groupe super créatif, on trouvé des manières d'utiliser les shaders (originalement conçu exclusivement pour des calculs liées à la synthèse d'images) afin d'implémenter des algorithmes plus générales toutes en prenant avantage de la puissance de calcul parallèle des GPUs

- Éventuellement, l'industrie a répondu en dessinant des architectures de GPU plus générales et des APIs cibler pour le GPGPU: CUDA, OpenCL, etc.

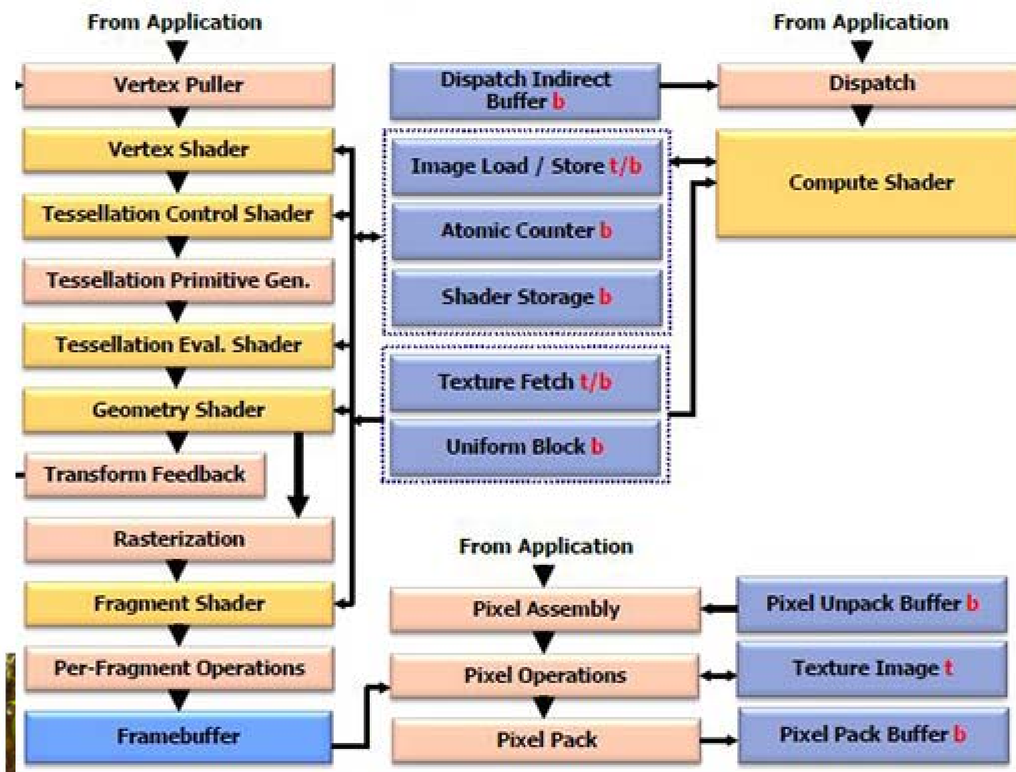
Université de Montréal

Évolution & extensions: rastérisation

- Ces APIs sont <u>excellents</u> si vous voulez seulement coder un algorithme parallèle et le faire tourner sur un GPU
  - Le moment que tu voudras intégrer ton algorithme de GPGPU avec un composant infographique (ex: visualisation d'un système de particules simuler avec le GPGPU), la communication entre l'API GPGPU et celle du pipeline rastérisation devenait un bottleneck
- Ce problème de bottleneck a plus ou moins été régler, mais il a aussi forcer les gens à se demander: est-ce qu'on a vraiment besoin de des APIs séparés pour l'infographie sur le GPU et le GPGPU?

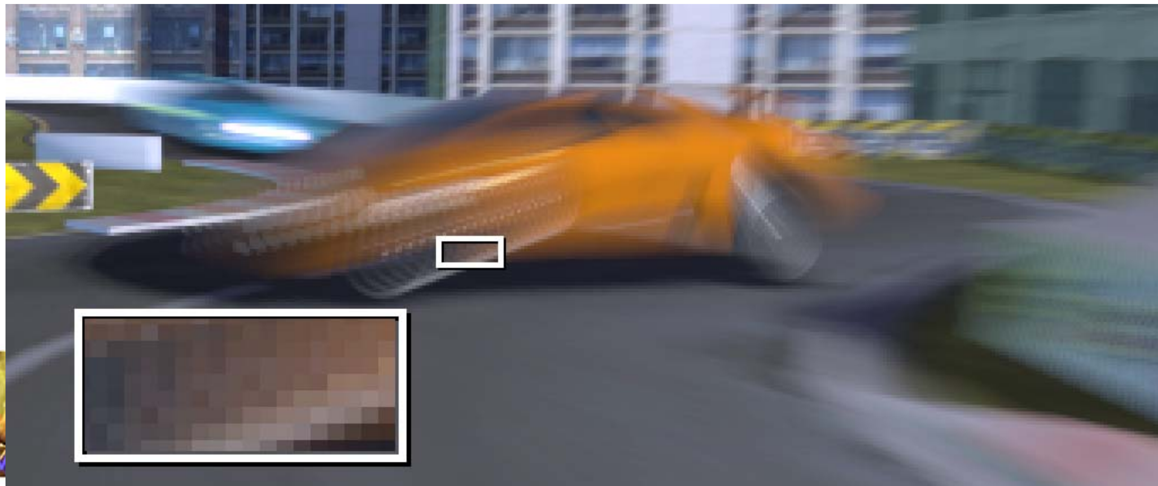
Évolution & extensions: rastérisation

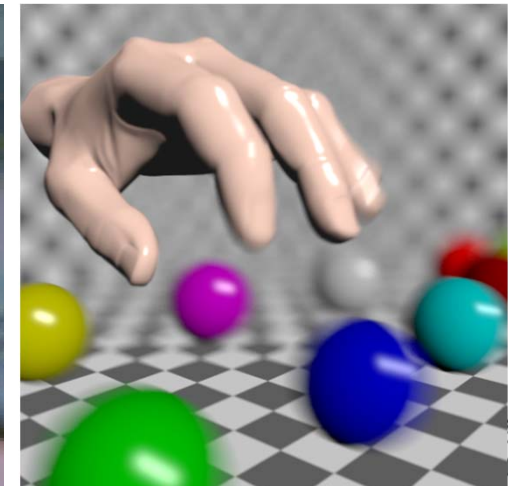
- Alors, les dernières versions de OpenGL et DirectX support un nouveau composant programmable: les *compute shaders*
  - Ils permettent le développement d'algorithmes générales* sur le GPU tout avec un API unifier au cas où ces algorithmes seront combinés avec des algorithmes de rendu



Le future: rastérisation

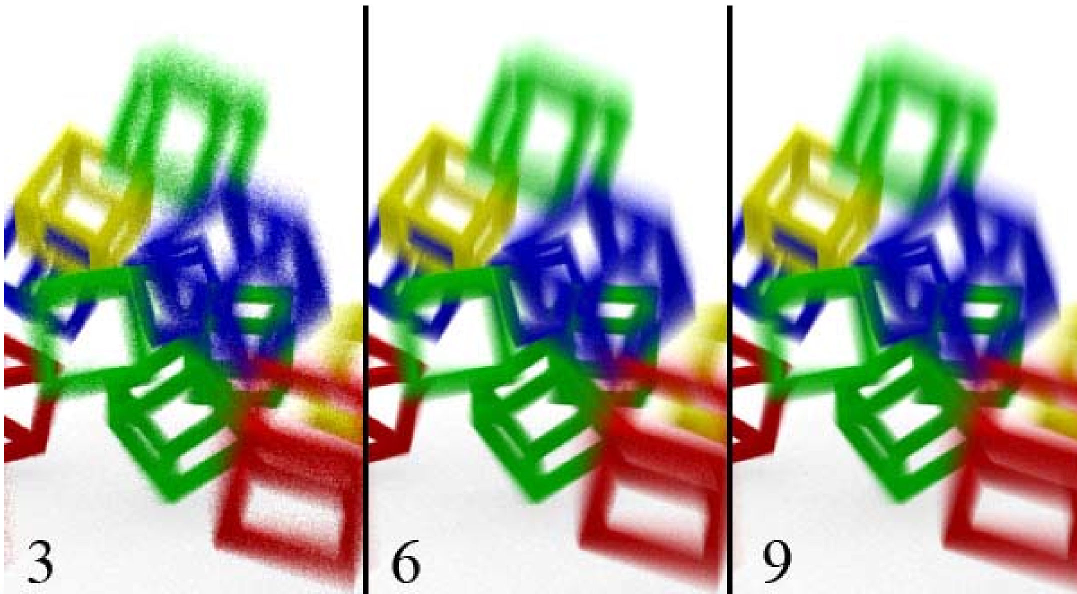
- Drôlement, tout les systèmes ce rapprochent
- Le future de rastérisation commence à suivre:
  - L'intégration d'un manière fondamentale des effets de distribution avec la *rastérisation stochastique*
  - L'utilisation des structures hiérarchiques pour gérer des scènes plus complexes ainsi que les calculs de visibilité
  - Rendre les calculs incohérent plus cohérent
- Ne risque pas de disparaître dans les prochaines 10 ans

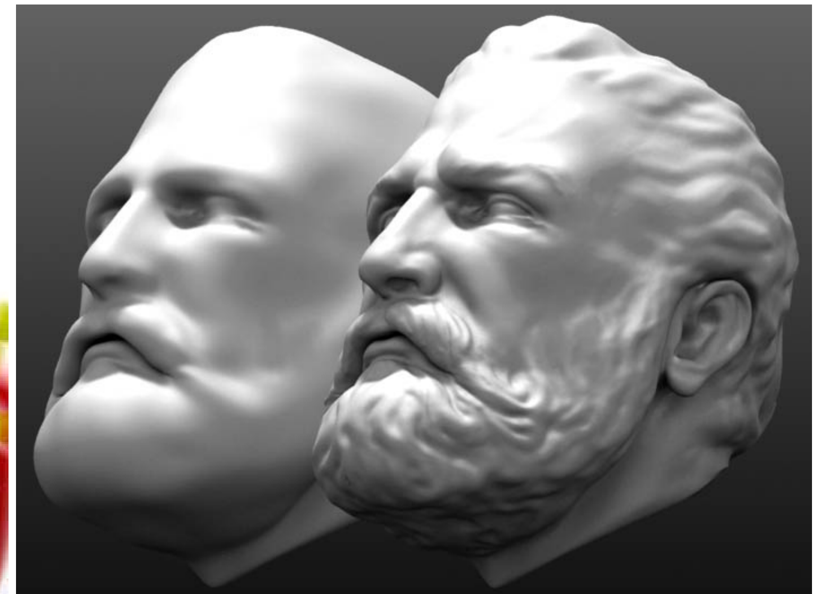




Évolution & extensions: micropolygon

- Comme les buts originaux de l’architecture de rastérisation micropolygon visaient la complexité et la réalisme, il n’est pas surprenant qu’elle a évolué selon ses axes…





Université de Montréal

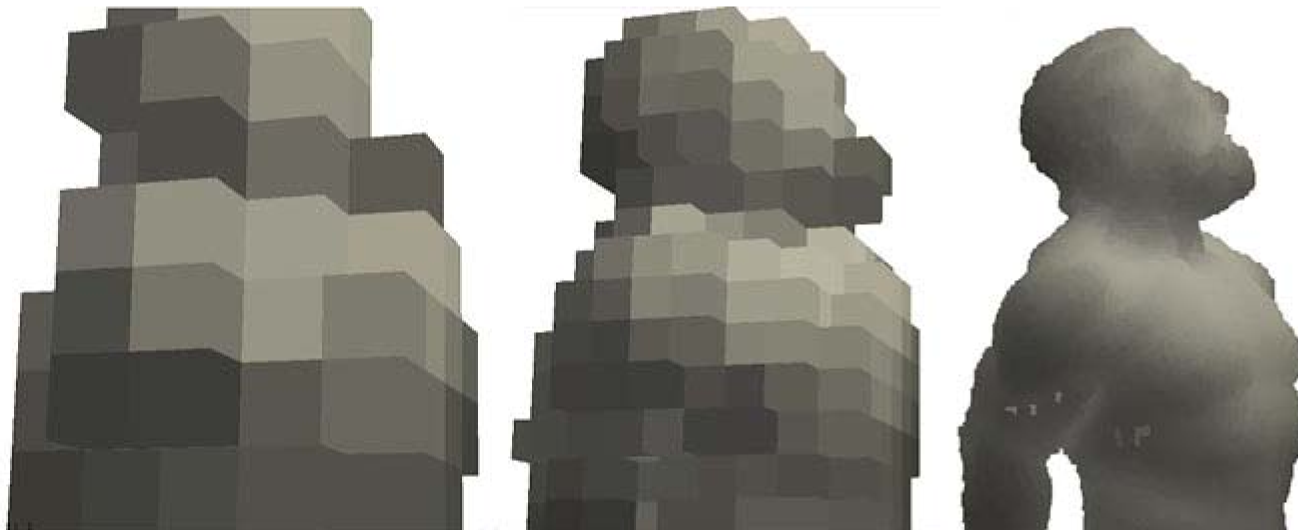
Évolution & extensions: micropolygon

- Afin de gérer des scènes de plus en plus complexes d'une manière adaptif, un des premiers extensions de PRMan était un représentation hiérarchique d'illumination et de géométrie, le *brickmap*, conçu selon deux observations:
  - le montant de détails nécessaires au loin est réduits, et
  - les effets distribués peuvent échantillonner (avec un taux réduit) selon des représentations filtrés

Université de Montréal

Évolution & extensions: micropolygon

- Afin de gérer des scènes de plus en plus complexes d'une manière adaptif, un des premiers extensions de PRMan était un représentation hiérarchique d'illumination et de géométrie, le *brickmap*, conçu selon deux observations:
  - le montant de détails nécessaires au loin est réduits, et
  - les effets distribués peuvent échantillonner (avec un taux réduit) selon des représentations filtrés



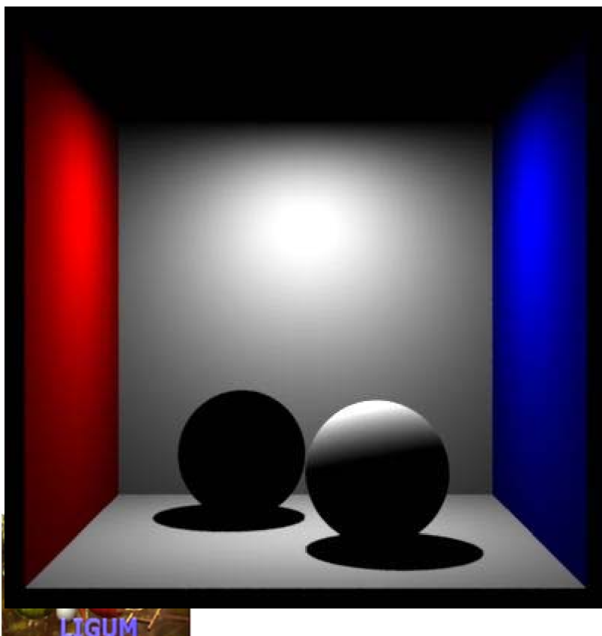
Université de Montréal

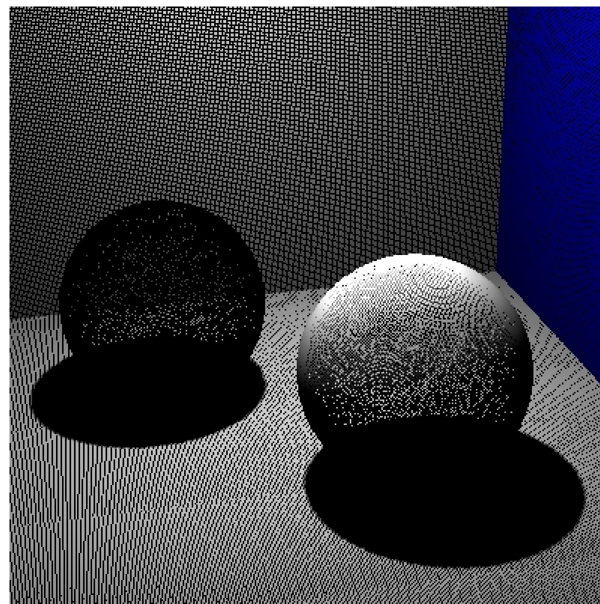
Évolution & extensions: micropolygon

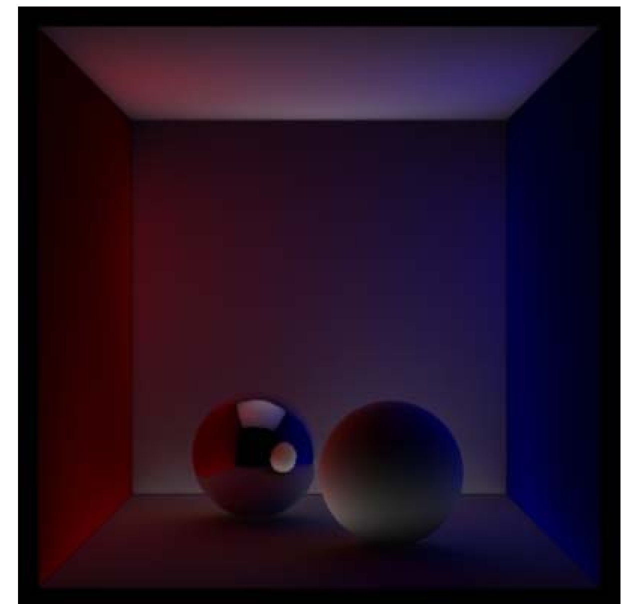
- Afin de gérer des scènes de plus en plus complexes d'une manière adaptif, un des premiers extensions de PRMan était un représentation hiérarchique d'illumination et de géométrie, le *brickmap*, conçu selon deux observations:
  - le montant de détails nécessaires au loin est réduits, et
  - les effets distribués peuvent échantillonner (avec un taux réduit) selon des représentations filtrés

Évolution & extensions: micropolygon

- Similairement, une représentation de point peuvent être utilisée d’une manière hiérarchique afin de sélectionner un niveau de détail suffisant pour le calcul de différent types d’effets







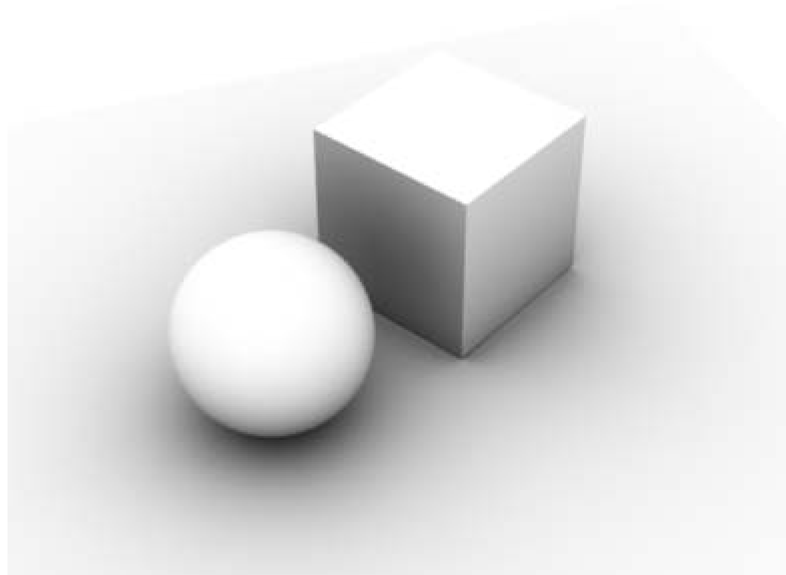
Évolution & extensions: micropolygon

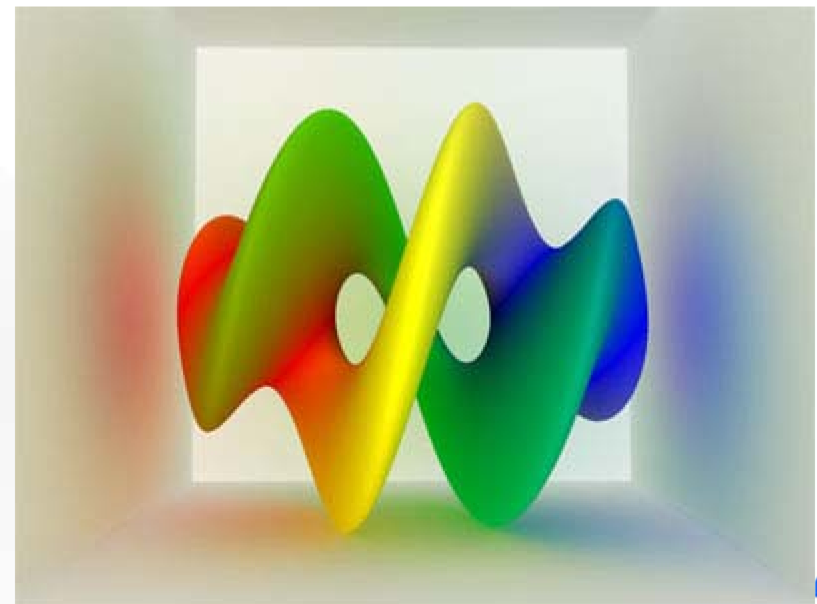
- Similairement, une représentation de point peuvent être utilisée d'une manière hiérarchique afin de sélectionner un niveau de détail suffisant pour le calcul de différent types d'effets

Université de Montréal

Évolution & extensions: micropolygon

- Similairement, une représentation de point peuvent être utilisée d’une manière hiérarchique afin de sélectionner un niveau de détail suffisant pour le calcul de différent types d’effets

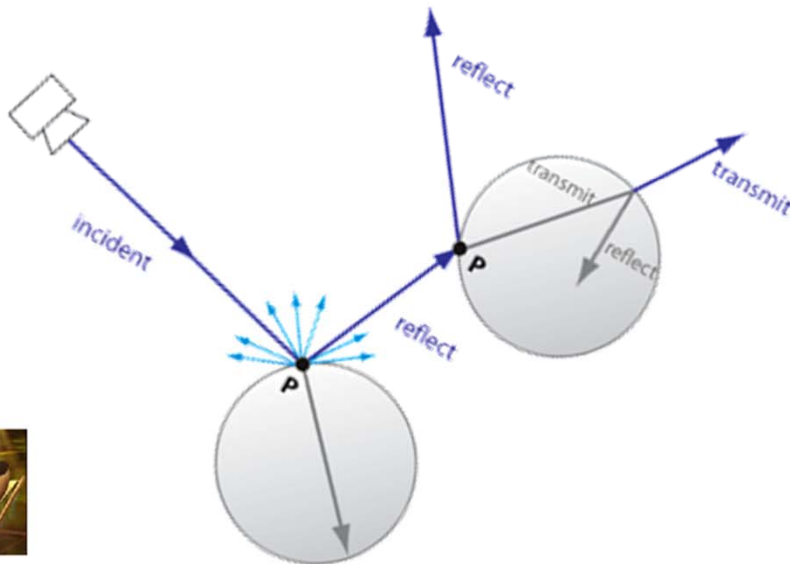


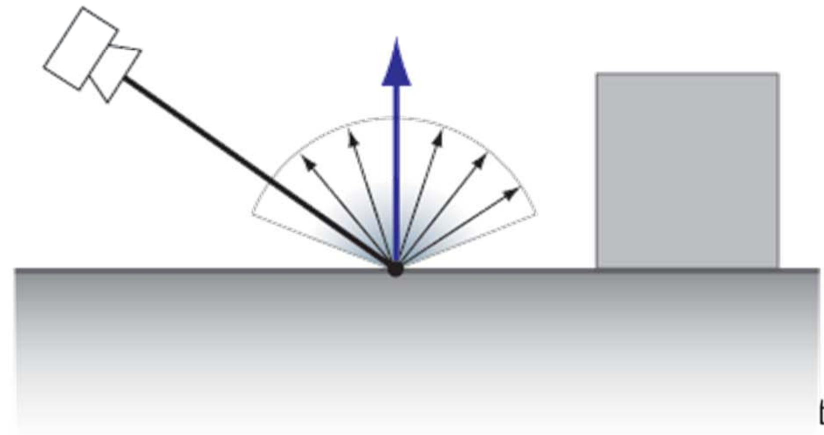


de Montréal

Évolution & extensions: micropolygon

- L'intégration d'un moteur de lancer dans PRMan a permis la simulation des effets plus réaliste
  - Pour (continuer à) gérer les scènes complexes, sans toujours respecter une exactitude mathématique, les représentations brickmap et PB sont aussi supporter dans le cadre de ce sous-système de lancer de rayons

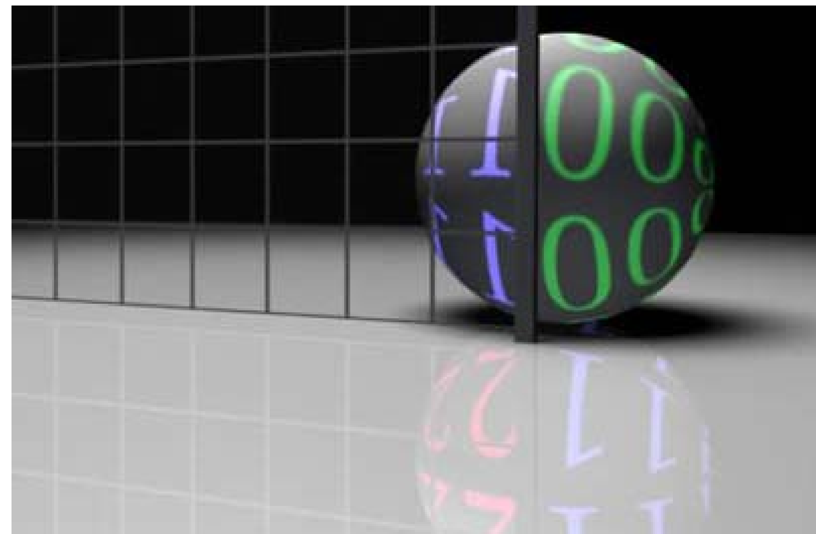




tréal

Évolution & extensions: micropolygon

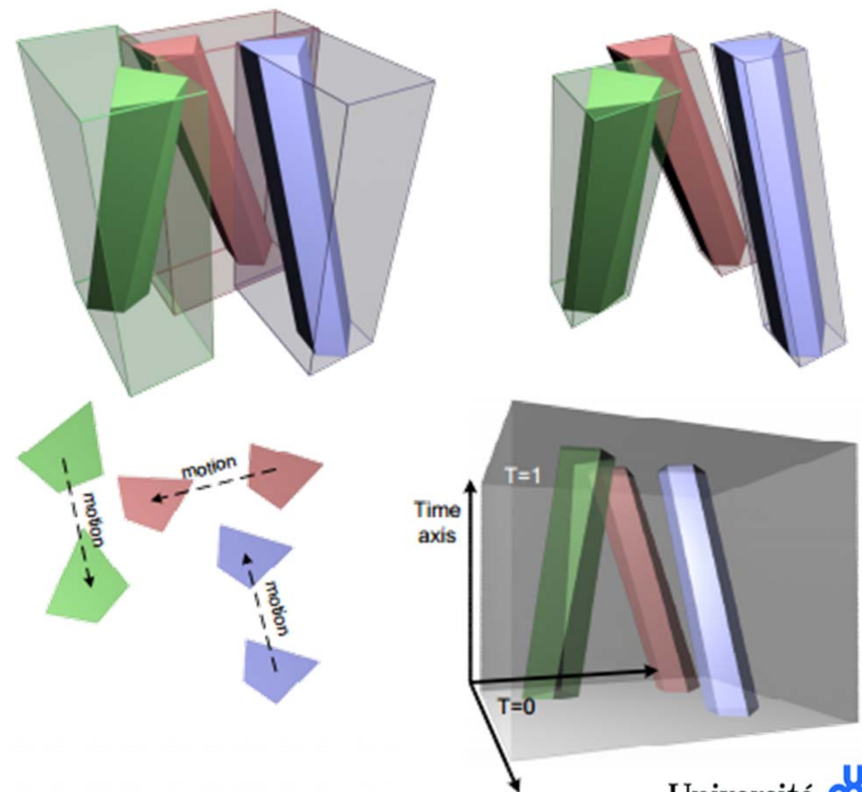
- L'intégration d'un moteur de lancer dans PRMan a permis la simulation des effets plus réaliste
  - Pour (continuer à) gérer les scènes complexes, sans toujours respecter une exactitude mathématique, les représentations brickmap et PB sont aussi supporter dans le cadre de ce sous-système de lancer de rayons



ersité
Montréal

Évolution & extensions: micropolygon

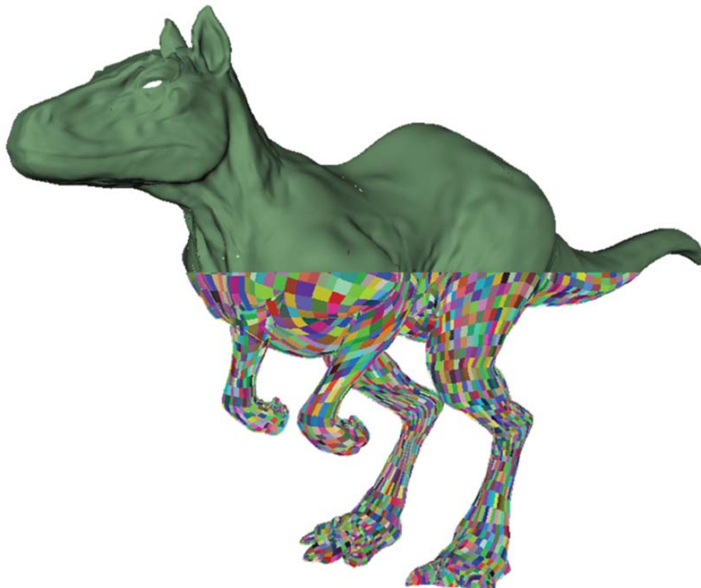
- Il est important de maintenir le support pour chaque effet avec chaque sous-système



Université de Montréal

Évolution & extensions: micropolygon

- Implémentation GPU
  - Il est très important d'avoir des résultats uniformes à travers les implémentations



Comment choisir un système

- Modèle d'illumination locale ou globale?
  - Besoin de quel type d'information de la scène à chaque élément de calcul?
  - Parallélisassions?
- Cohérence (ou non) en "espace rayons"
- Quel type de représentation géométrique
- Est-ce que vous avez besoin: d'éfficacité, de précision, tous les deux?
- Il devient de plus en plus fréquent qu'une combinaison de système/représentation sont nécessaires pour atteindre une haute performance

Université de Montréal

Survol / révision / référence: techniques de rendu…

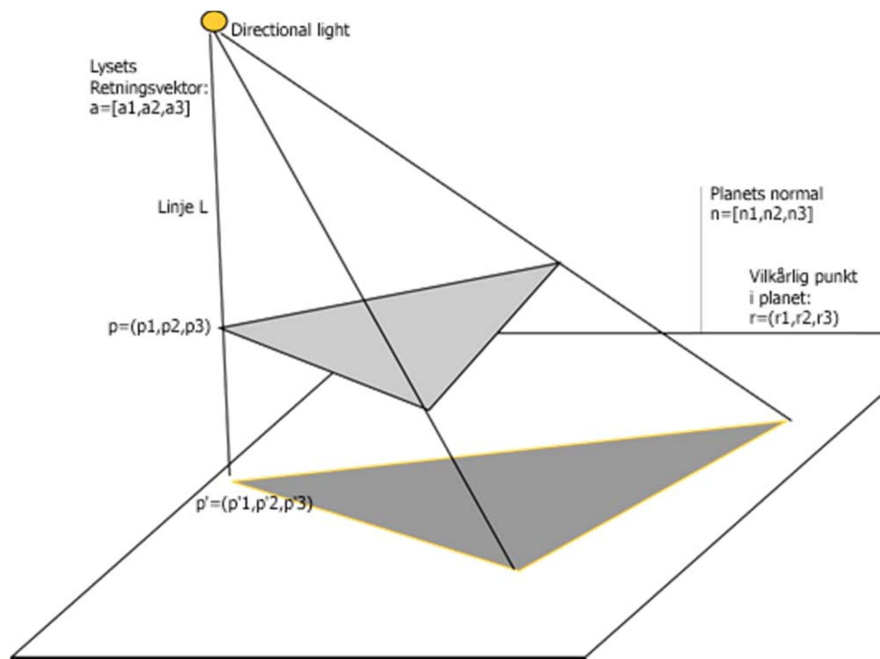
Techniques: Illumination directe

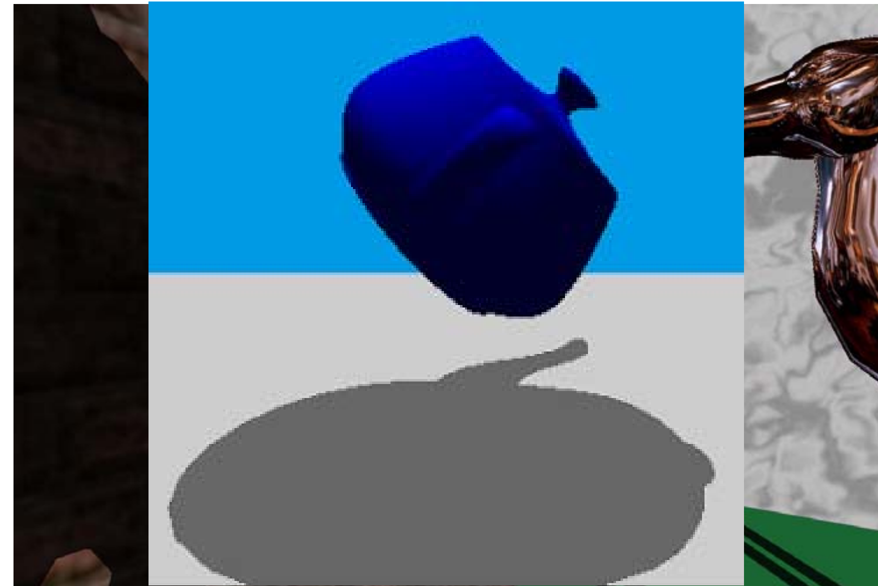
- *Illuminations et matériaux simples*
  – Lancer de rayons, ombres projectives, *shadow mapping*, volumes d'ombre (*shadow volumes*)

Université de Montréal

Techniques: Illumination directe

- *Illuminations et matériaux simples*
  - Lancer de rayons, ombres projectives, *shadow mapping*, volumes d'ombre (*shadow volumes*)

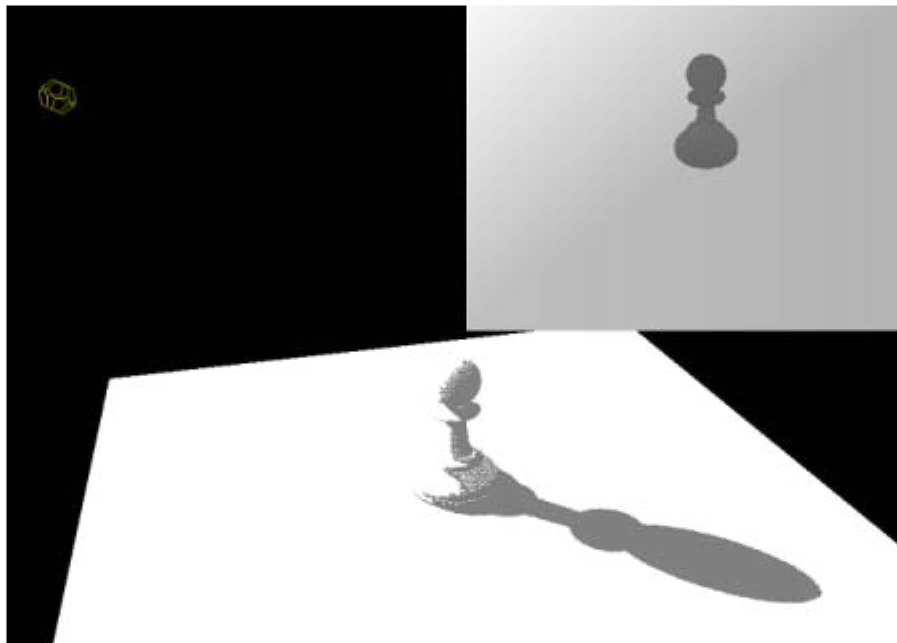


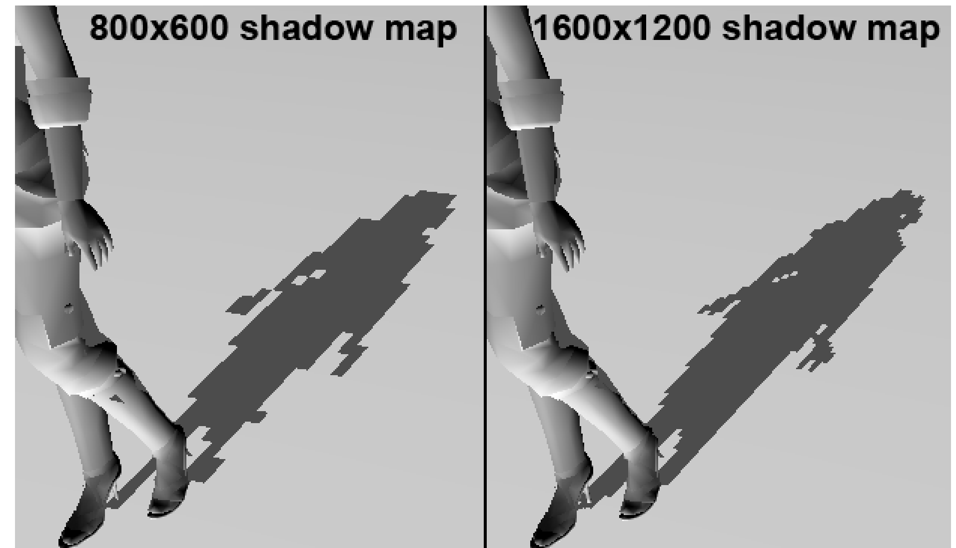


Université de Montréal

Techniques: Illumination directe

- *Illuminations et matériaux simples*
  - Lancer de rayons, ombres projectives, *shadow mapping*, volumes d'ombre (*shadow volumes*)

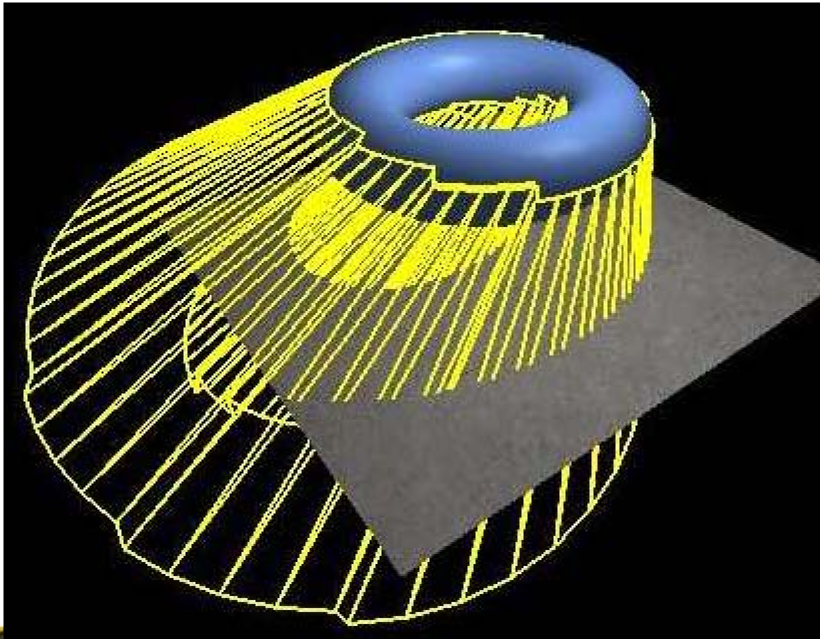




Université de Montréal

Techniques: Illumination directe

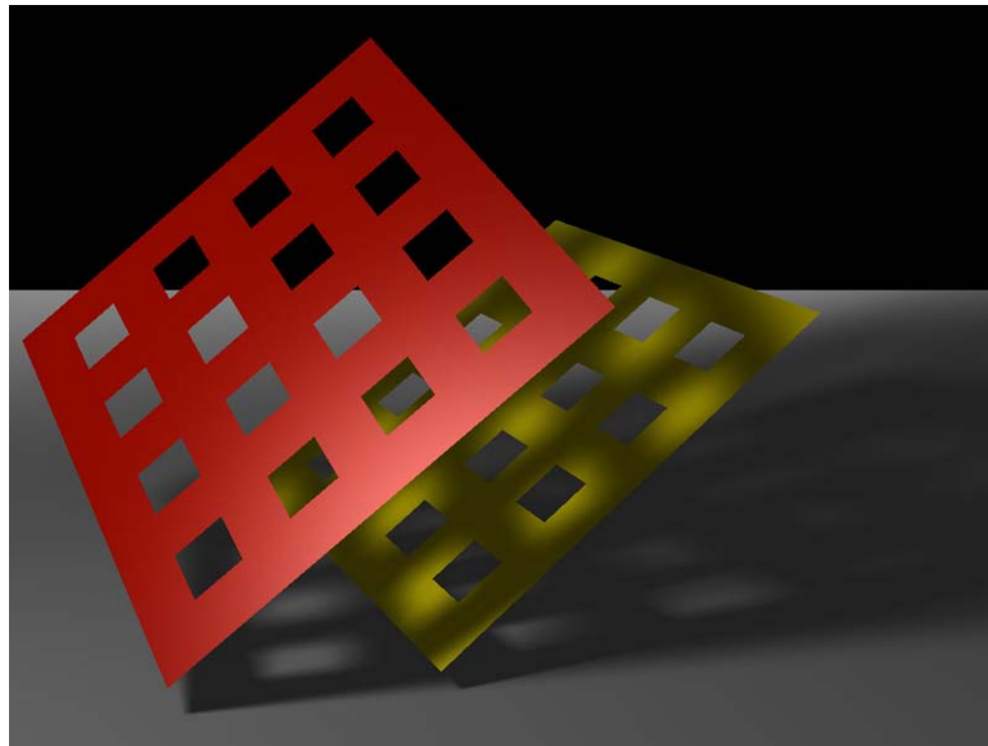
- *Illuminations et matériaux simples*
  - Lancer de rayons, ombres projectives, *shadow mapping*, volumes d'ombre (*shadow volumes*)



Université de Montréal

Techniques: Illumination directe

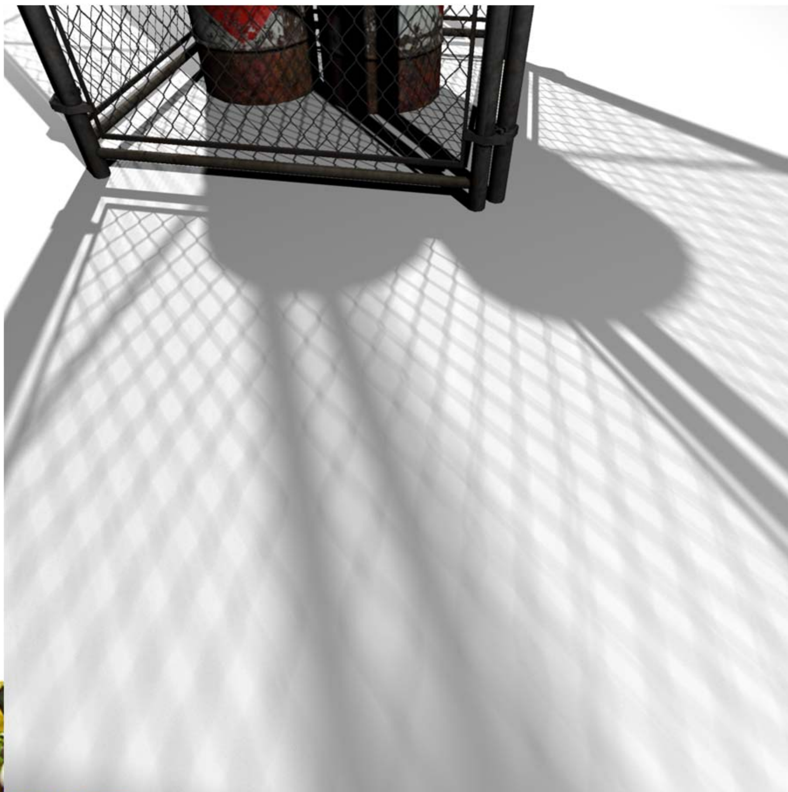
- *Lumières étendues*
  - Lancer de rayons, *soft shadow volumes*, intégration de silhouette

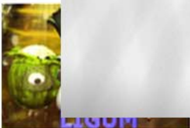


Université de Montréal

Techniques: Illumination directe

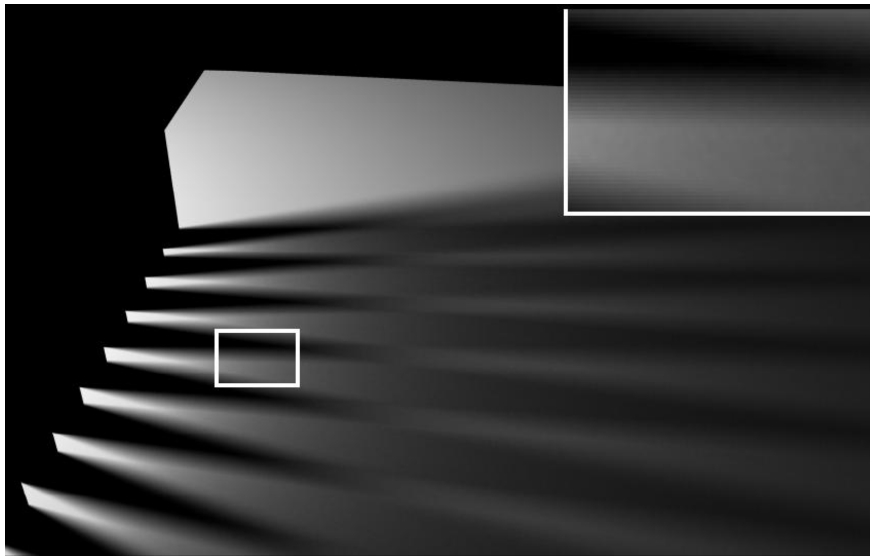
- *Lumières étendues*
  - Lancer de rayons, *soft shadow volumes*, intégration de silhouette





Techniques: Illumination directe

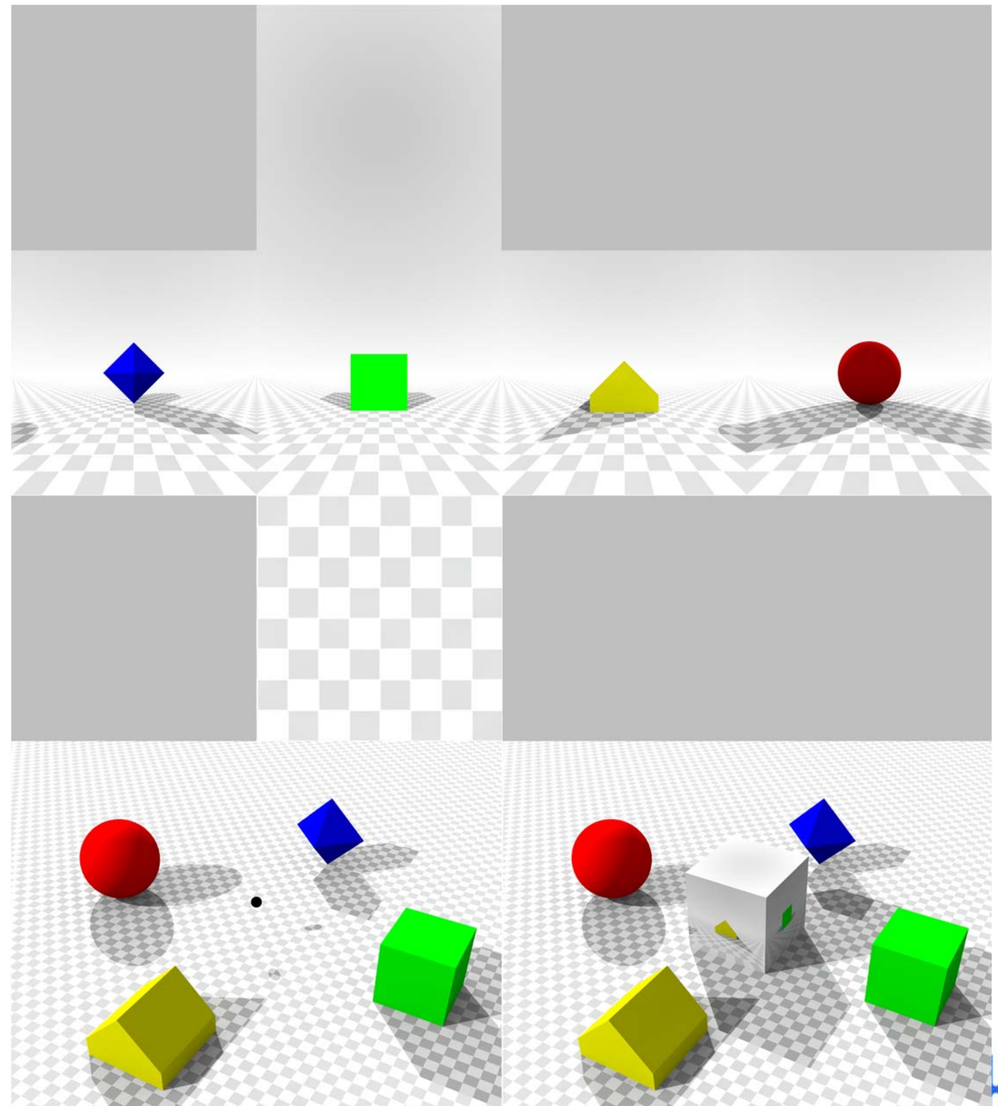
- *Lumières étendues*
  - Lancer de rayons, *soft shadow volumes*, intégration de silhouette



Université de Montréal

Techniques: Illumination directe

- *Réflexion complexe*
  - Lancer de rayons, réflexion projective, *reflection mapping*, etc…



Université de Montréal

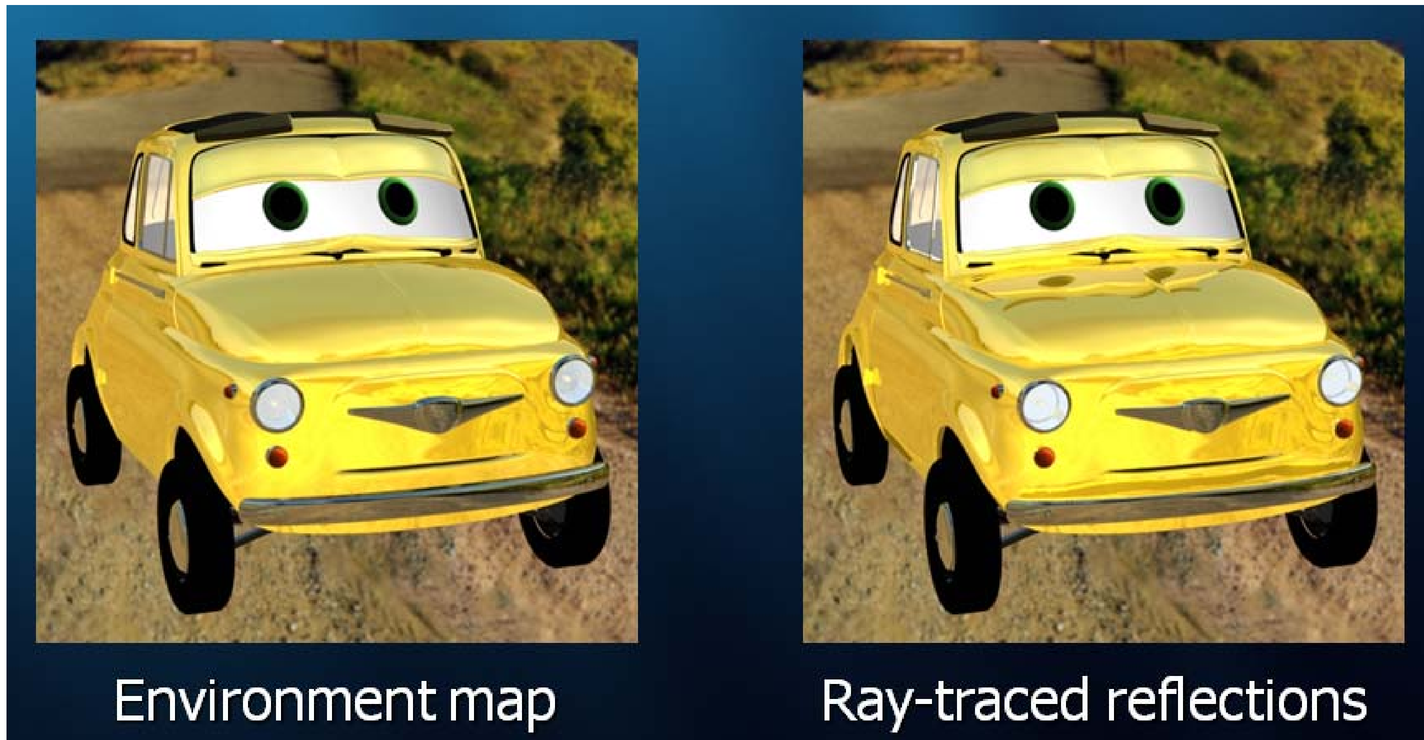
Techniques: Illumination directe

- *Réflexion complexe*
  - Lancer de rayons, réflexion projective, *reflection mapping*, etc…

Université de Montréal

Techniques: Illumination directe

- *Réflexion complexe*
  - Lancer de rayons, réflexion projective, *reflection mapping*, etc…



Université de Montréal

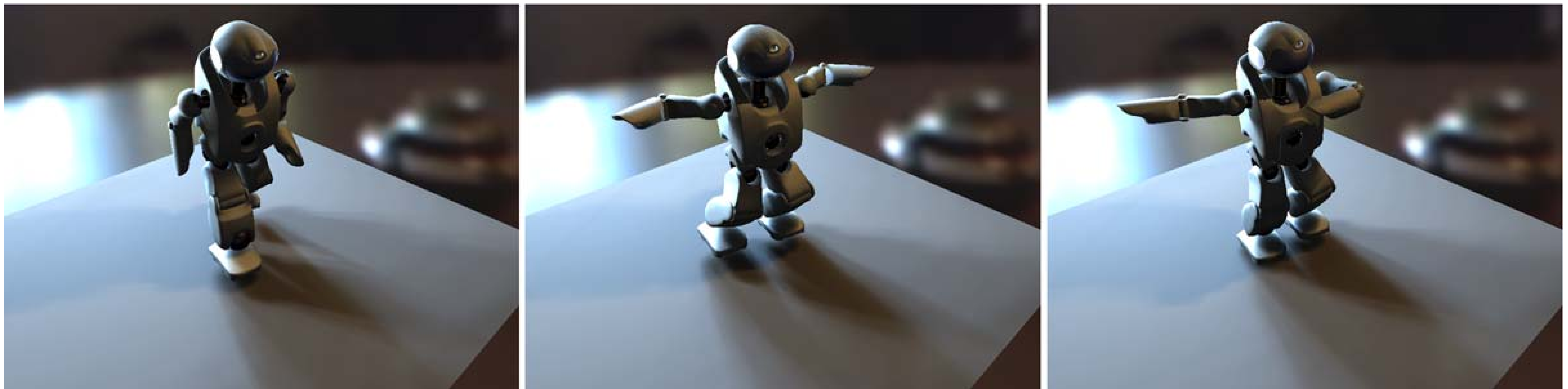
Techniques: Illumination directe

- *Lumières environnementales*
  - Lancer de rayons, *shadow maps*, *precomputed radiance transfer*

Université de Montréal

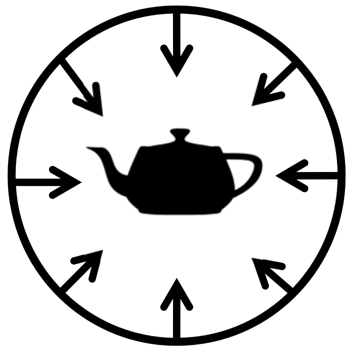
Techniques: Illumination directe

- *Lumières environnementales*
  - Lancer de rayons, *shadow maps*, *precomputed radiance transfer*



Techniques: Illumination directe

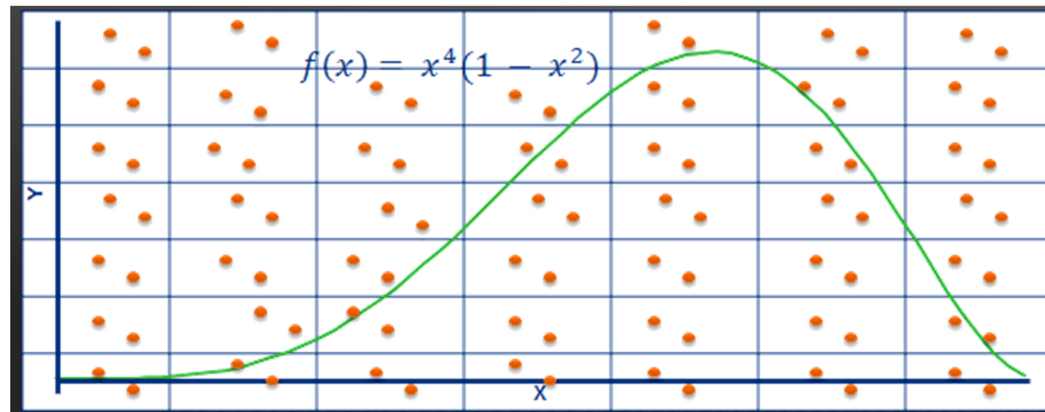
- *Lumières environnementales*
  - Lancer de rayons, *shadow maps*, *precomputed radiance transfer*

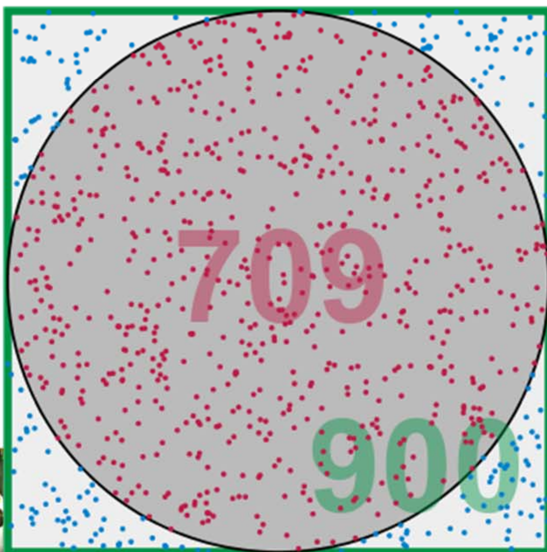


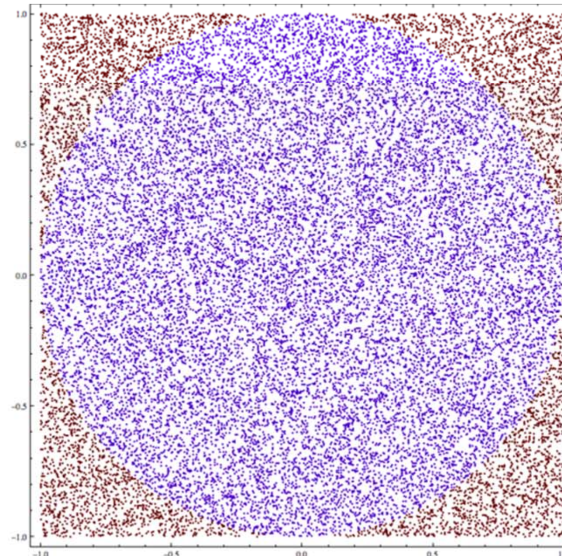
Université de Montréal

Techniques: Illumination globale

- *Intégration Monte Carlo*



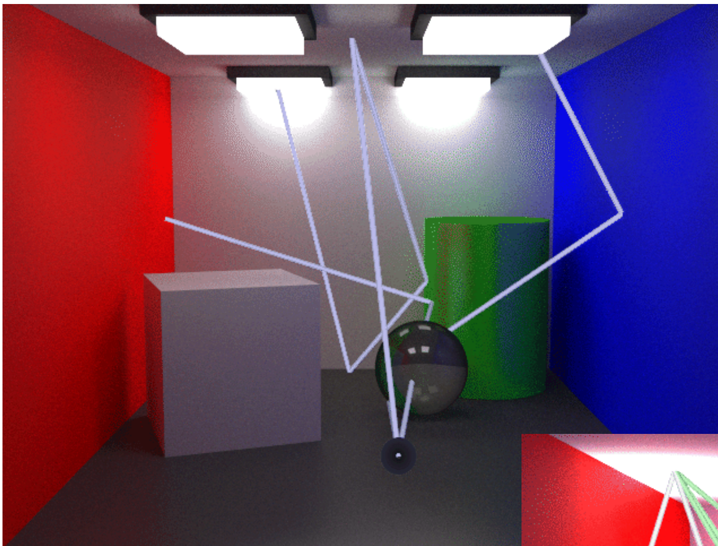


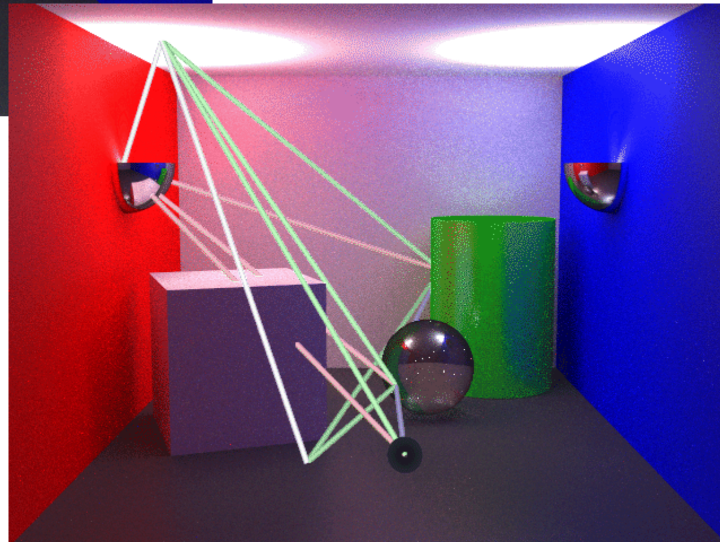


de Montréal

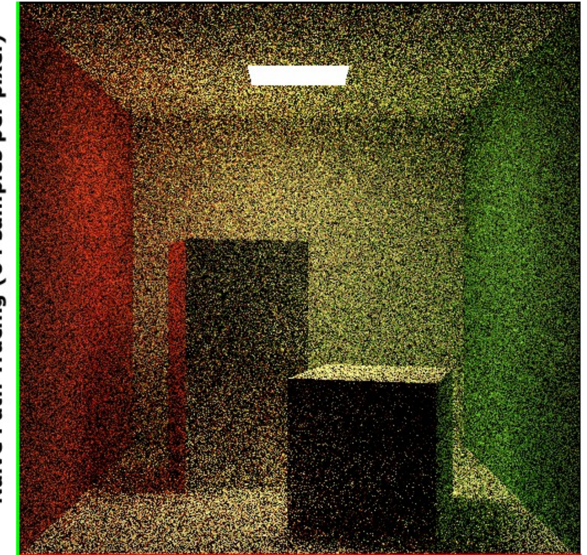
Techniques: Illumination globale

- *Path tracing*





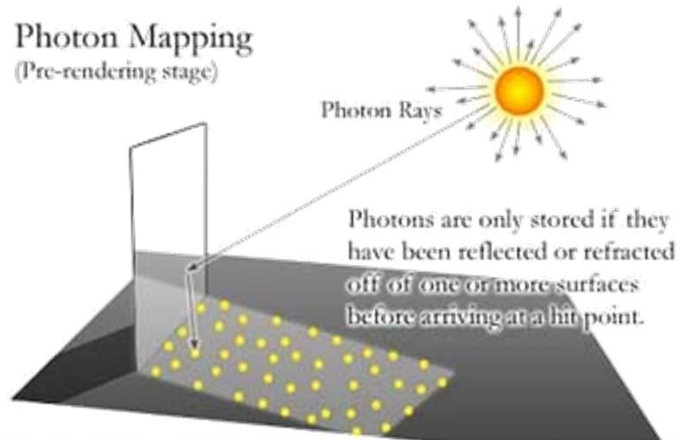
Naive Path Tracing (64 samples per pixel)

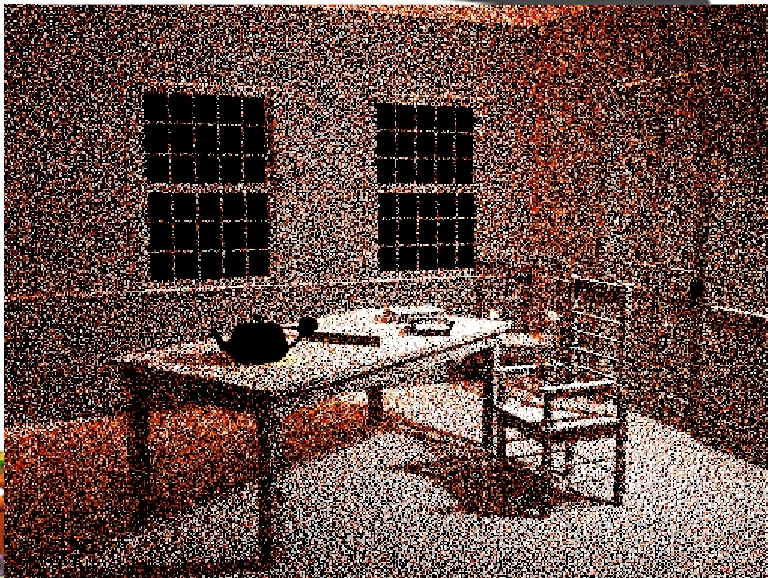


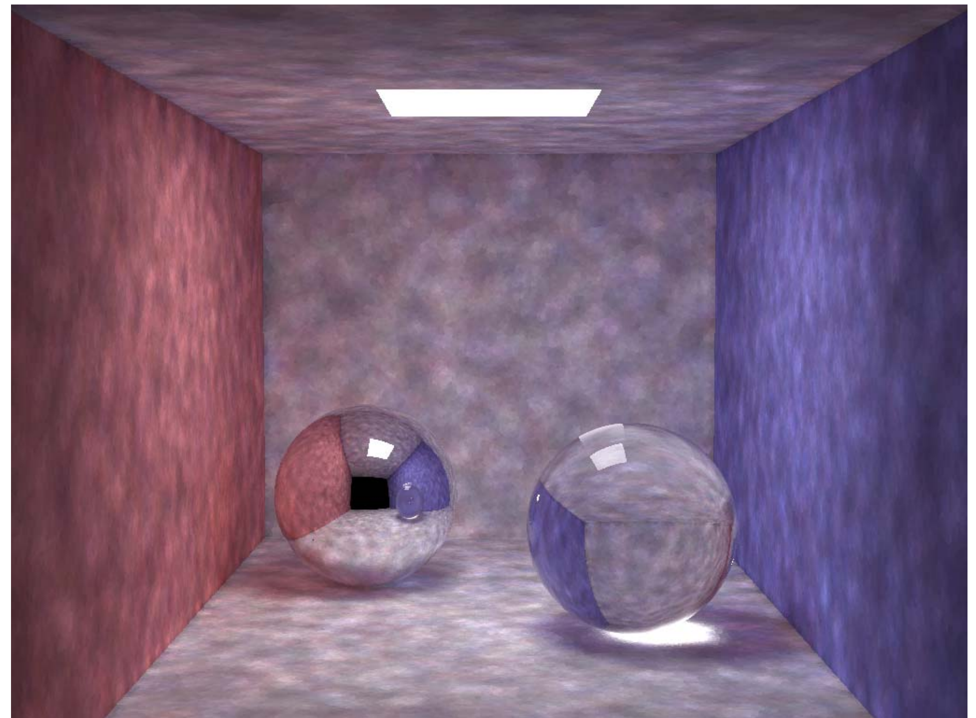
Path Tracing With Explicit Direct (64 samples)

Techniques: Illumination globale

- *Photon mapping*

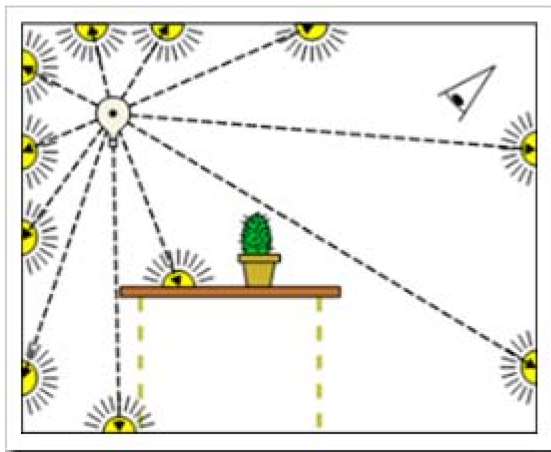


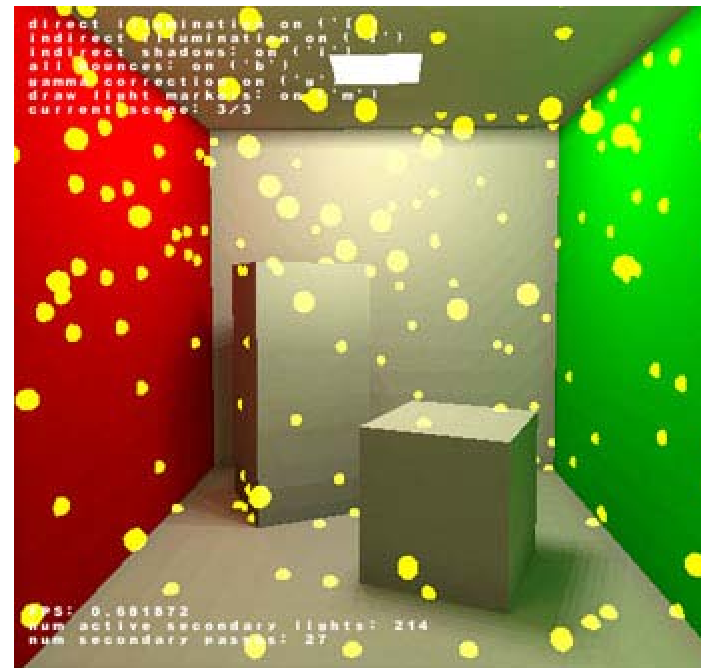




Techniques: Illumination globale

- *Virtual Lights*
  - traitement des singularités faibles, *reflective shadow maps, imperfect shadow maps, light cuts*

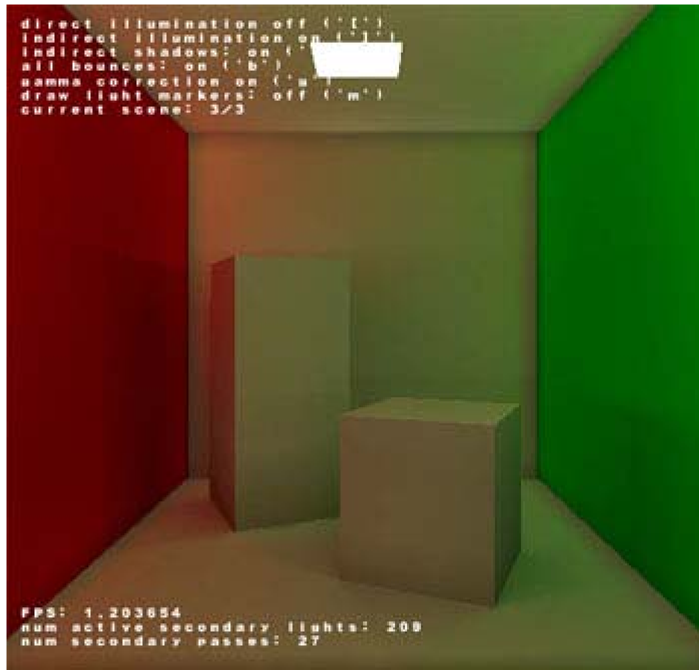


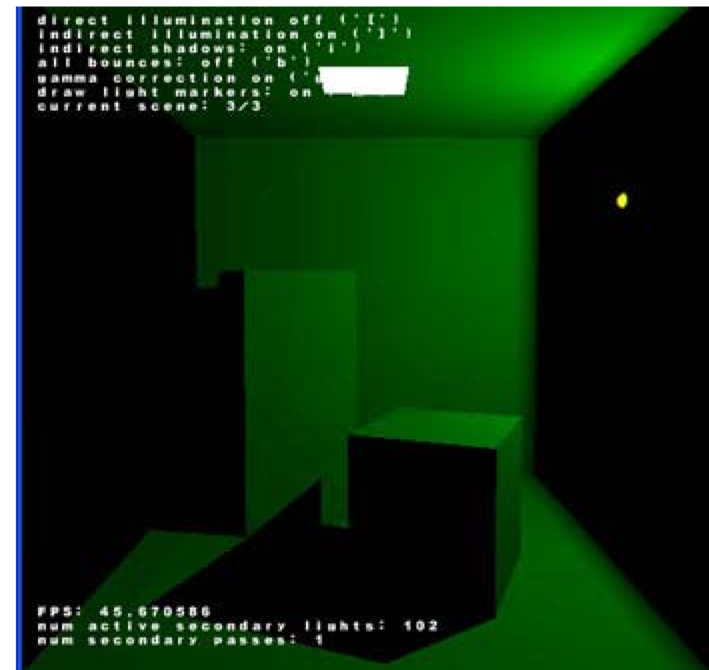


Université de Montréal

Techniques: Illumination globale

- *Virtual Lights*
  - traitement des singularités faibles, *reflective shadow maps, imperfect shadow maps, light cuts*

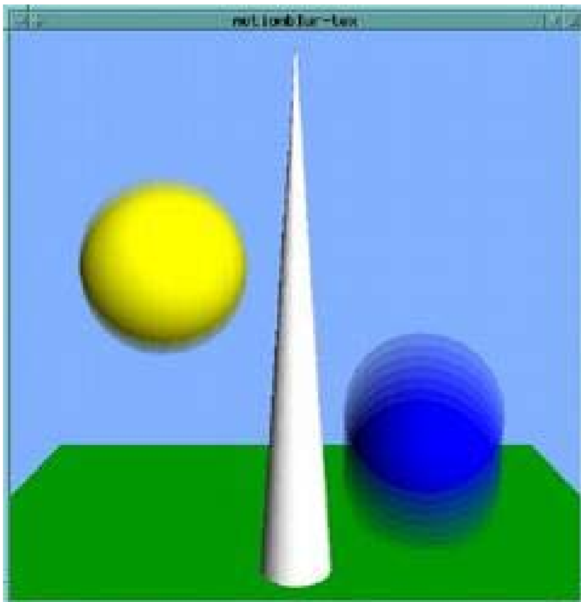


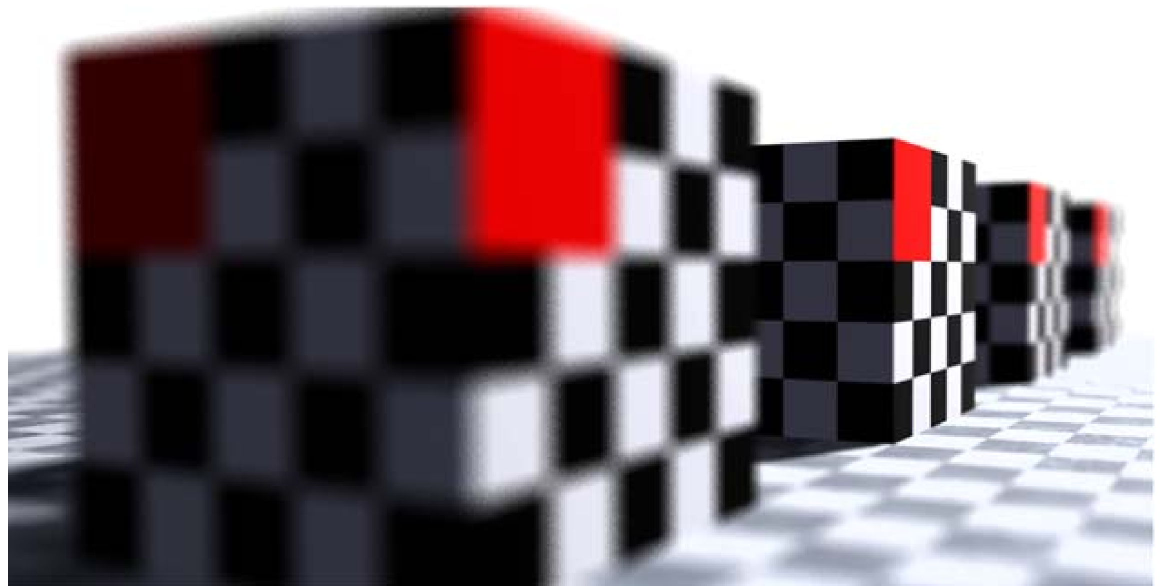


Université de Montréal

Effets spéciaux / distribués

- *Accumulation buffer,* rastérisation stochastique, techniques *screen-space*, etc…

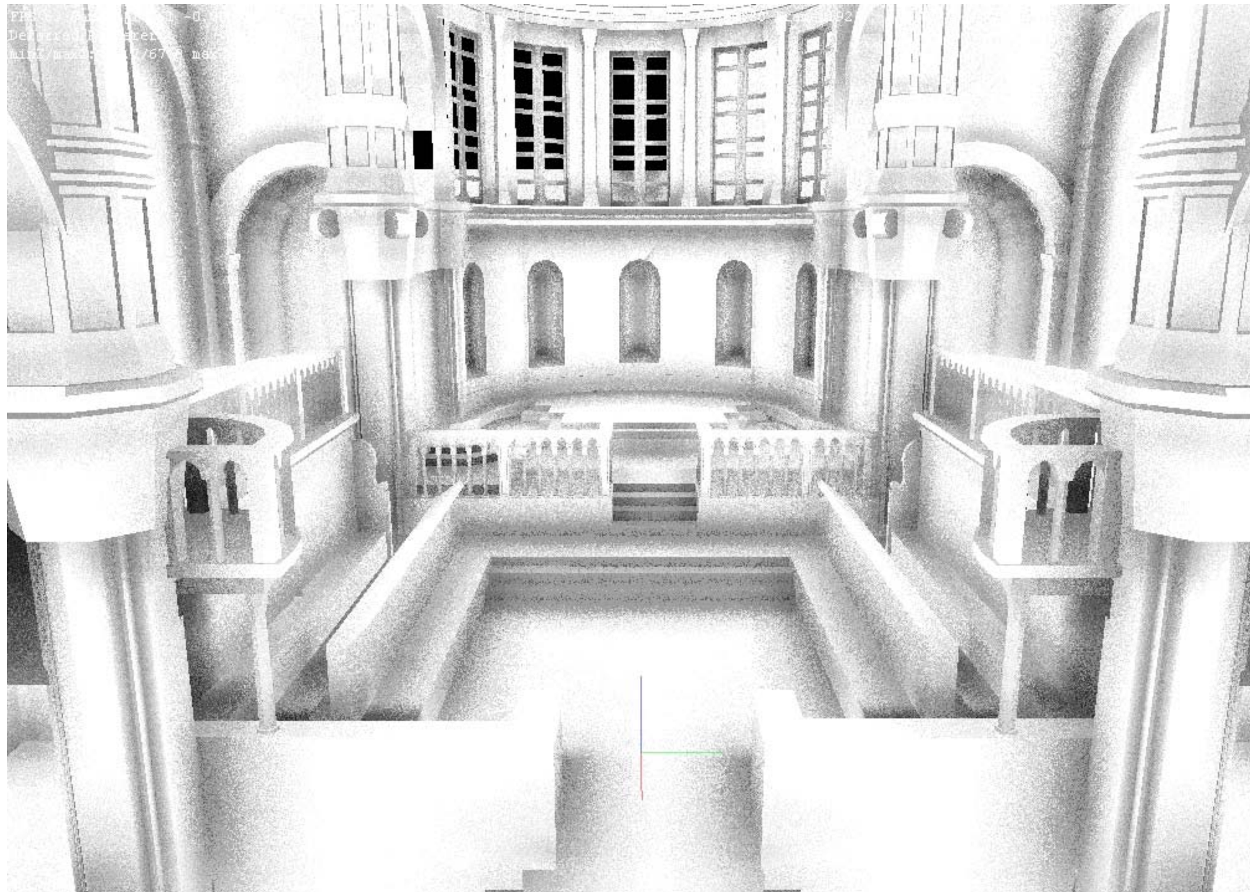




Université de Montréal

Effets spéciaux / distribués

- *Accumulation buffer,* rastérisation stochastique, techniques *screen-space*, etc…



Université de Montréal

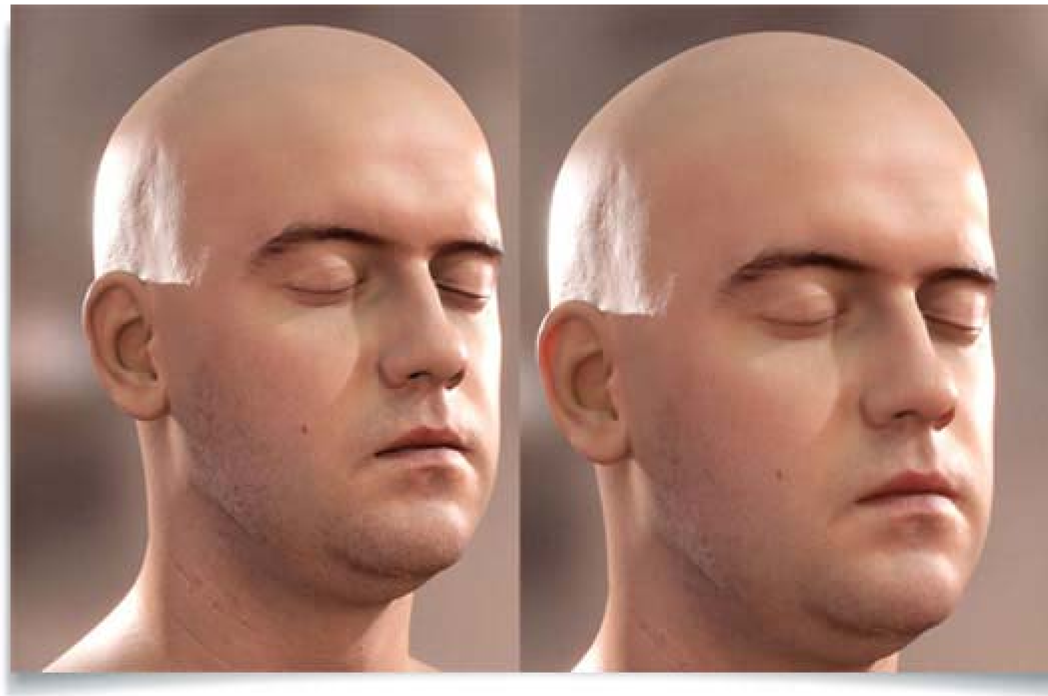
Effets spéciaux / distribués

- *Accumulation buffer,* rastérisation stochastique, techniques *screen-space*, etc…

Université de Montréal

Effets spéciaux / distribués

- *Accumulation buffer,* rastérisation stochastique, techniques *screen-space*, etc…



Université de Montréal