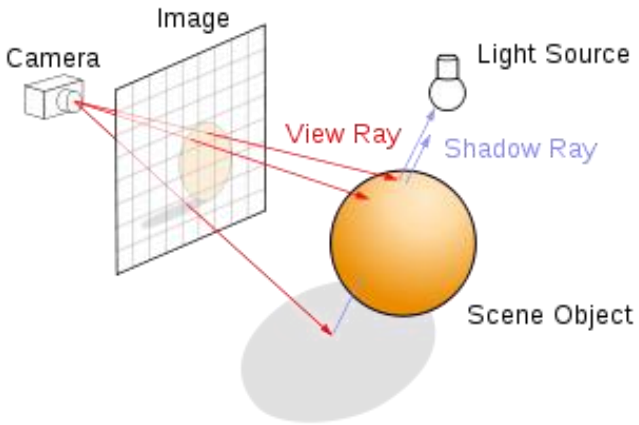
Chapitre 6.1 – Le *ray tracer* en infographie

Un *ray tracer* en quelques mots

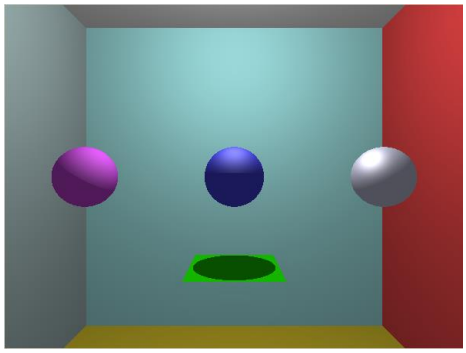
Un *ray tracer* est un logiciel permettant de visualiser une scène modélisée en 3D fondé sur la théorie de l'optique géométrique inversée. L'objectif est de **lancer des rayons** depuis un point d'observation (une caméra) et de suivre la **trajectoire optique inverse**[1] de ce rayon.



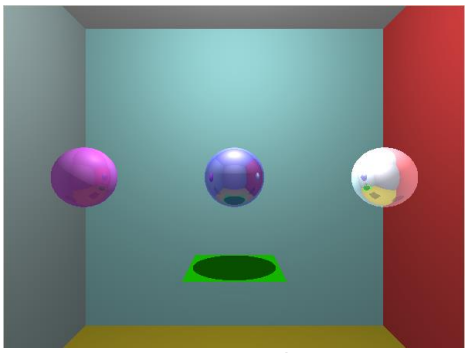
http://en.wikipedia.org/wiki/Ray_tracing_(graphics)
Description visuelle du fonctionnement d'un *ray tracer*.

Lorsqu'il y a une **intersection** entre le rayon et un objet de l'environnement, un calcul d'**illumination** (*shading*) détermine si l'objet intersecté est éclairé par une source de lumière et évalue la couleur que celui-ci va réfléchir et transmettre vers le point d'observation.

Les résultats dépendent de la qualité de l'algorithme et du détail de la scène :



Illumination directe seulement avec source de lumière blanche au-dessus de la sphère bleue.



Illumination directe et indirecte à un niveau avec des sphères peu et très miroir. Une source de lumière blanche est au-dessus de la sphère bleue.

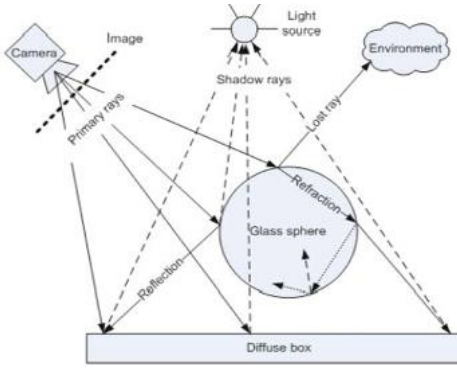
Termes techniques d'un *ray tracer*

Scène

Une scène représente un environnement en trois dimensions où l'on retrouve des objets (primitives) que l'on désire visualiser à l'aide d'une image en deux dimensions.

Caméra

La caméra représente le point de référence par rapport auquel on effectue la visualisation de la scène.



http://oivdoc90.vsg3d.com/content/513-what-real-time-ray-tracing
Illustration de plusieurs rayons lancés depuis la caméra par un *ray tracer*.

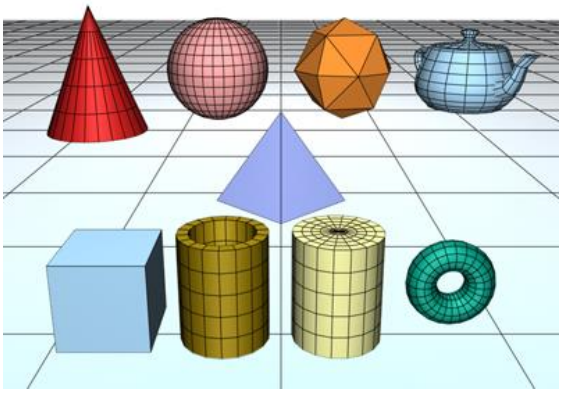
[1] Le trajet réel de la lumière est toujours de la source de lumière à l'œil (ou la caméra).

Rayon

Un rayon correspond à la trajectoire inverse de la lumière initialement partant de la caméra. L'intersection de ce rayon avec un objet de la scène permettra de déterminer la couleur de l'image à générer.

Géométrie

Une géométrie représente la forme d'un objet (plan, sphère, triangle, cube, etc).



http://www.sluniverse.com/php/vb/general-sl-discussion/68454-parametric-tools-2.html
Illustration de géométries typiques.

Matériel

Un matériel représente les caractéristiques physiques et visuelles d'un objet (couleur, indice de réfraction, transparence, etc).

Primitive (objet de la scène)

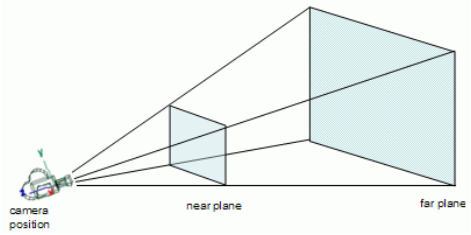
Une primitive est un objet de la scène que l'on caractérise par une géométrie et un matériel.

Source de lumière

Une source de lumière permet d'illuminer une scène afin de rendre visible les matériaux des primitives dont la géométrie a été intersectée par un rayon. Une source de lumière peut être bloquée par des géométries ce qui reproduit l'effet d'ombrage.

Pyramide de vue (*view frustum*)

Une pyramide de vue correspond à la région de la scène qui sera vue par la caméra. Elle est délimitée par un écran de face (*near clipping plane*) et un écran de fond (*far clipping plane*).



http://www.lighthouse3d.com/tutorials/view-frustum-culling/
Pyramide de vue associée à une caméra.

Algorithme d'illumination (*shader*)

Un algorithme d'illumination est un calcul évaluant la **réflexion** et la **transmission** d'une source de lumière sur une primitive. Ce calcul dépend de plusieurs facteurs comme la géométrie et le matériel de la primitive éclairée.

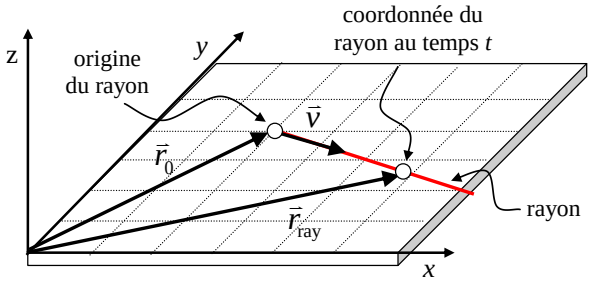
http://knowledge.autodesk.com/support/3ds-max/learn-explore/caas/CloudHelp/cloudhelp/2015/ENU/3DSMax/files/GUID-4D919F24-F0BB-47F0-9C2D-28D393C1A1DE-htm.html
Scène avec un rendu avec un algorithme d'illumination de haute qualité.

Écran de vue (*viewport*)

Un écran de vue est une grille régulière de pixels. L'objectif du *raytracer* sera de lancer un rayon depuis la caméra dans chaque pixel afin de calculer une couleur qui dépendra de l'intersection du rayon avec les différentes primitives de la scène et de leur illumination.

L'équation d'un rayon

Un rayon $\vec{r}_{\text{ray}}$ est une structure ayant une origine $\vec{r}_0$ et une orientation $\vec{v}$. Il permet de faire des calculs d'intersection avec des formes géométriques d'une scène. On utilise un paramètre de temps *t* pour ordonnée chronologique les intersections du rayon avec les objets de la scène.



Un rayon paramétré dans le temps aura la forme suivante :

$$\vec{r}_{\text{ray}} = \vec{r}_0 + \vec{v}t$$

où $\vec{r}_{\text{ray}}$: Coordonnée touchée par le rayon après un temps *t*.

$\vec{r}_0$: Origine du rayon.

$\vec{v}$: Orientation du rayon ($\|\vec{v}\| = 1$, vecteur unitaire).

t : Temps écoulé dans le déplacement du rayon.

En informatique, la définition d'un rayon aura besoin des variables suivantes :

Géométrie du rayon	Information sur la géométrie intersectée
• Position d'origine du rayon ($\vec{r}_0$) • Orientation du rayon ($\vec{v}$)	• Le temps *t* pour intercepter la géométrie. • La normale à la surface $\vec{n}$ au site de l'interception. • Une coordonnée de texture *uv* (s'il y en a une). • Référence vers le matériel appliqué sur la géométrie (ex : pour obtenir la couleur de la géométrie).

La caméra

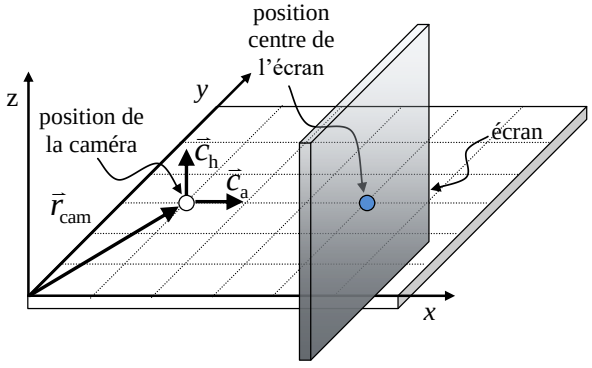
La caméra représente le site à partir du quel nous allons prendre une image en deux dimensions d'une scène en trois dimension. Pour ce faire, il nous faut la position de la caméra, son orientation et la définition du « haut » de la caméra.

<u>Paramètres de la caméra :</u>

$\vec{r}_{\text{cam}}$: Position de la caméra.

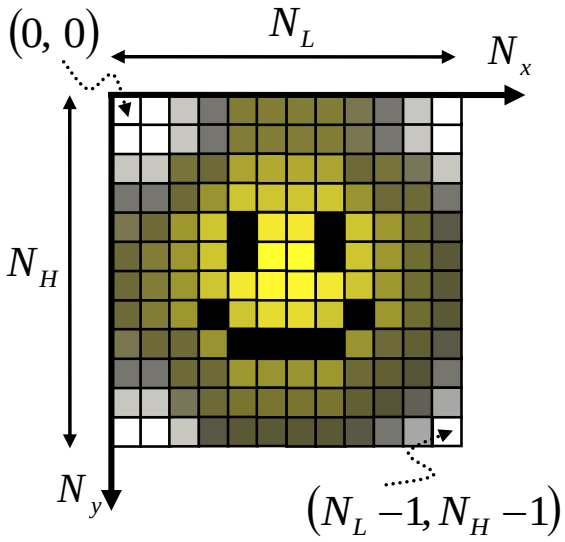
$\vec{c}_{\text{a}}$: Orientation <u>l'avant</u> de la caméra ($\|\vec{c}_{\text{a}}\| = 1$, vecteur unitaire).

$\vec{c}_{\text{h}}$: Orientation du <u>haut</u> de la caméra ($\|\vec{c}_{\text{h}}\| = 1$, vecteur unitaire).



L'écran de vue (*viewport*)

L'écran de vue (*viewport*) correspond à la portion de la scène qui sera visible par la caméra. Cet écran est composé d'une grille régulière à deux dimensions dont chaque élément porte le nom de « pixel ». Il y a N_L pixels en largeur (*width*) et N_H pixels en hauteur (*height*). La largeur n'est pas nécessairement égale à la hauteur. De nos jours, la résolution d'un moniteur[2] est de l'ordre de 1440 × 900.



En infographie, on fait correspondre un **axe *x*** de nombre entier N_x avec la **largeur** dont le sens **positif** est **vers la droite** et un **axe *y*** de nombre entier N_y avec la **hauteur** avec un **sens positif vers le bas**. Le pixel de coordonnée $(N_x = 0, N_y = 0)$ est alors situé dans le coin supérieur gauche de l'écran (voir schéma ci-contre).

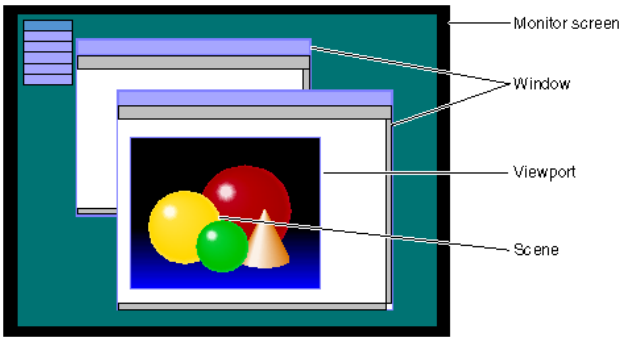
Afin d'affecter la bonne couleur à chaque pixel de l'écran de vue, il faudra lancer un rayon $\vec{r}_{ray}$ depuis la position de la caméra $\vec{r}_{cam}$ dans chaque pixel (N_x , N_y) de l'écran de vue. Il faudra faire correspondre à chaque pixel (N_x , N_y) une coordonnée (x, y, z) dans l'espace monde (*world space*) de la scène 3d tel que

$$(N_x , N_y) \rightarrow \vec{r}_0 = (x, y, z)$$

afin de déterminer la position $\vec{r}_0$ d'entrée du rayon $\vec{r}_{ray}$ dans la scène (puisque ce rayon n'est pas officiellement lancé depuis la caméra). Ce calcul de transformation d'espace sera réalisé par la pyramide de vue (*view frustum*).

Afin de clarifier certains termes techniques, voici une représentation visuelle de plusieurs éléments se retrouvant dans la description de l'écran :

- moniteur (*monitor screen*)
- fenêtre d'un programme (*window*)
- écran de vue (*viewport*)
- scène 3d à visualiser (*scene*)

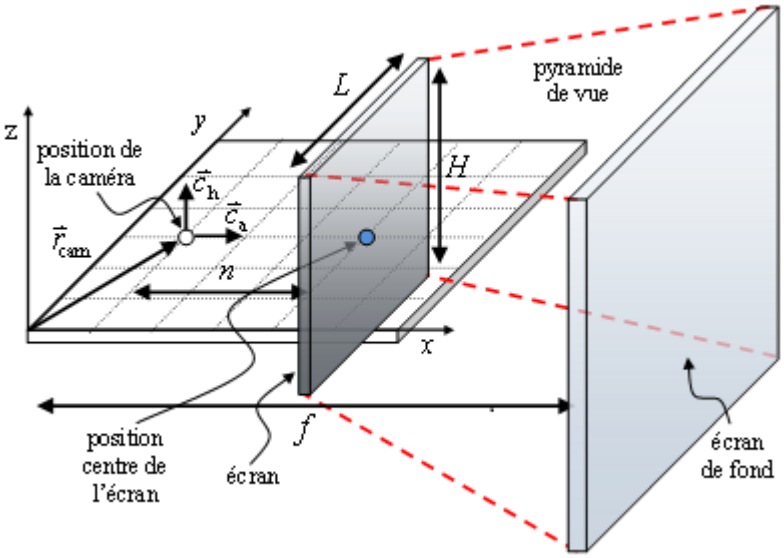


http://techpubs.sgi.com/library/tpl/cgi-bin/getdoc.cgi?coll=nt&db=bks&srch=&fname=/SGI_Developer/Cos3C_PG/sgi_html/ch09.html

[2] Pour un écran de 19''.

La pyramide de vue (*view frustum*)

La pyramide de vue (*view frustum*) correspond à la région de la scène qui sera visualisée par la caméra. L'écran de face (*near clipping plane*) représente une surface de taille $L \times H$ dans l'espace *xyz* de la scène sur laquelle la visualisation de la scène va s'effectuer. D'une certaine façon, elle représente l'écran de vue dans l'espace de la scène. La profondeur de la pyramide de vue est limitée par l'écran de fond (*far clipping plane*).



Pour bien définir la pyramide de vue, il faut préciser la distance n entre la caméra et l'écran de face et la distance f entre la caméra et l'écran de fond.

Définition des paramètres :

n : Distance entre la caméra et l'écran de face (*near clipping plane*).

f : Distance entre la caméra et l'écran de fond (*far clipping plane*).

L : Largeur de l'écran (*width*).

N_L : Nombre de pixels en largeur de l'écran de vue (*viewport*).

H : Hauteur de l'écran (*height*).

N_H : Nombre de pixels en hauteur de l'écran de vue (*viewport*).

Comment déterminer *L* et *H* adéquatement

Afin de déterminer le plus judicieusement la largeur L et la hauteur H de l'écran de face, il est préférable d'utiliser un angle d'ouverture θ selon l'axe vertical et d'utiliser un ratio a basé sur le nombre de pixels N_L en largeur et N_H en hauteur afin de produire une image sans distorsion. Le nombre de pixels disponibles dépend de la fenêtre du rendu (*viewport*) utilisée dans l'application.

Habituellement, on utilise un angle d'ouverture tel que

$$\theta = 60°$$

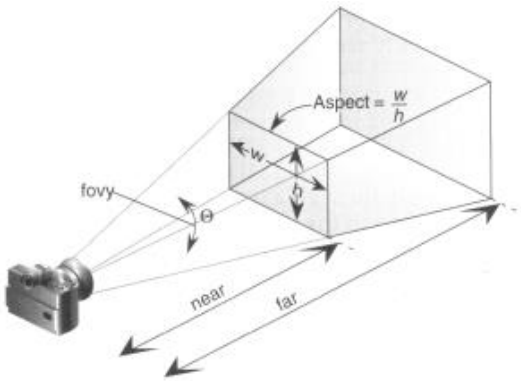


Figure 3-14 Perspective Viewing Volume Specified by gluPerspective()

https://andrewharvey4.wordpress.com/2009/12/

N_L ou w : Nombre de pixels sur la largeur.

N_H ou h : Nombre de pixels sur la hauteur.

C'est ce type d'implémentation que la librairie *OpenGl* propose d'utiliser grâce à la fonction *gluPerspective*[3] pour configurer la pyramide de vue dans le calcul du rendu :

```
void gluPerspective(GLdouble fovy,
                    GLdouble aspect,
                    GLdouble zNear,
                    GLdouble zFar);
```

`fovy`
Specifies the field of view angle, in degrees, in the *y* direction.

`aspect`
Specifies the aspect ratio that determines the field of view in the *x* direction. The aspect ratio is the ratio of *x* (width) to *y* (height).

`zNear`
Specifies the distance from the viewer to the near clipping plane (always positive).

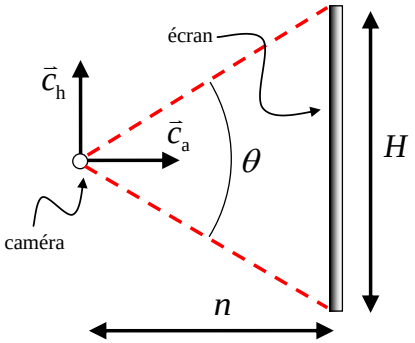
`zFar`
Specifies the distance from the viewer to the far clipping plane (always positive).

Pour définir adéquatement la hauteur H de l'écran à partir de n et θ, nous pouvons utiliser la relation trigonométrique

$$\tan\left(\frac{\theta}{2}\right) = \frac{H/2}{n}$$

ce qui nous donne

$$H = 2n\tan\left(\frac{\theta}{2}\right).$$



Pour éviter la distorsion de l'image, on utilise le ratio a à partir de N_L et N_H tel que

$$a = \frac{N_L}{N_H}$$

ce qui nous permet de définir la largeur L de l'écran à l'aide de l'équation

$$L = aH.$$

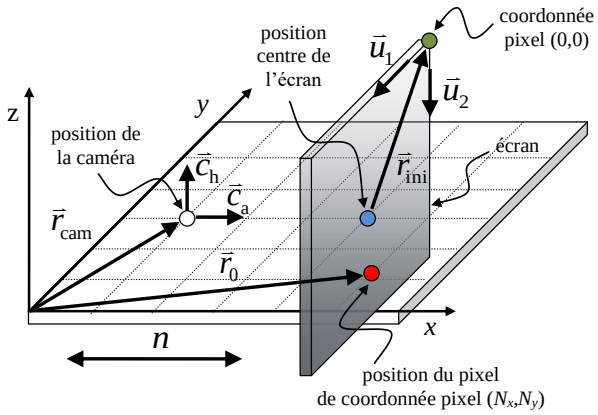
En résumé, nous pouvons déterminer L et H de la façon suivante :

Angle d'ouverture à définir	Ratio des pixels	Hauteur de l'écran	Largeur de l'écran
$\theta = 60°$ (habituellement)	$a = \frac{N_L}{N_H}$	$H = 2n\tan\left(\frac{\theta}{2}\right)$	$L = aH$

[3] http://www.opengl.org/sdk/docs/man2/xhtml/gluPerspective.xml

La localisation du pixel sur l'écran

Pour construire un rayon, on débute avec la localisation du pixel à colorer sur l'écran en utilisant deux vecteurs « pixel unitaire » $\vec{u}_1$ et $\vec{u}_2$. Ces deux vecteurs permettent se déplacer sur la surface de l'écran de face à coup de « taille de pixel ».



La position du pixel sera

$$\vec{r}_0 = \vec{r}_{\text{cam}} + n\vec{c}_{\text{a}} + \vec{r}_{\text{ini}} + N_x\vec{u}_1 + N_y\vec{u}_2$$

avec l'usage des vecteurs pixels unitaires

$$\vec{u}_1 = \frac{\vec{c}_{\text{a}} \times \vec{c}_{\text{h}}}{\left\|\vec{c}_{\text{a}} \times \vec{c}_{\text{h}}\right\|} \frac{L}{N_{\text{L}}} \qquad \text{et} \qquad \vec{u}_2 = \frac{\vec{c}_{\text{a}} \times \vec{u}_1}{\left\|\vec{c}_{\text{a}} \times \vec{u}_1\right\|} \frac{H}{N_{\text{H}}}$$

et l'usage du vecteur

$$\vec{r}_{\text{ini}} = -\frac{N_{\text{L}}}{2}\vec{u}_1 - \frac{N_{\text{H}}}{2}\vec{u}_2$$

localisant le pixel de coordonnée (0,0) situé dans le coin supérieur gauche de l'écran de vue depuis le centre de l'écran. Il faut préciser que $\vec{r}_0$ tel que défini donne une coordonnée (x,y,z) au pixel dans l'espace de scène associé au coin supérieur gauche du pixel (N_x, N_y). Au besoin, on peut **modifier l'équation** de $\vec{r}_0$ pour avoir comme coordonnée le **centre du pixel** ou tout simplement une **coordonnée aléatoire** dans le pixel.

Définition des paramètres :

$\vec{r}_0$: Position du pixel à colorer.

$\vec{r}_{\text{cam}}$: Position de la caméra.

$\vec{c}_{\text{a}}$: Orientation l'avant de la caméra ($\left\|\vec{c}_{\text{a}}\right\| = 1$, vecteur unitaire).

$\vec{c}_{\text{h}}$: Orientation du haut de la caméra ($\left\|\vec{c}_{\text{h}}\right\| = 1$, vecteur unitaire).

n : Distance entre la caméra et l'écran de projection en 2D de la scène.

$\vec{r}_{\text{ini}}$: Vecteur pour positionner le pixel de coordonnée (0,0) à partir du centre de l'écran.

N_x : Coordonnée écran du pixel selon l'axe largeur de l'écran ($N_x \in [0..\ N_{\text{L}} - 1]$, entier).

N_y : Coordonnée écran du pixel selon l'axe hauteur de l'écran ($N_y \in [0..\ N_{\text{H}} - 1]$, entier).

L : Largeur de l'écran (*width*).

N_{L} : Nombre de pixels en largeur.

H : Hauteur de l'écran (*height*).

N_{H} : Nombre de pixels en hauteur.

$\vec{u}_1$: Vecteur pixel unitaire qui est orienté selon l'axe largeur de l'écran vers la droite.

$\vec{u}_2$: Vecteur pixel unitaire qui est orienté selon l'axe hauteur de l'écran vers le bas.

La construction du rayon

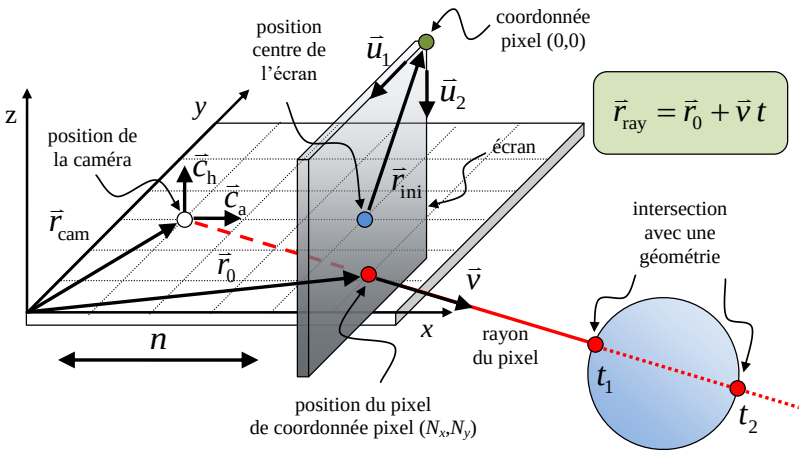
La construction d'un rayon $\vec{r}_{\text{ray}}$ nécessite la position du pixel $\vec{r}_0$ sur l'écran et l'orientation $\vec{v}$ du rayon que l'on définit comme étant

$$\vec{v} = \frac{\vec{r}_0 - \vec{r}_{\text{cam}}}{\left\|\vec{r}_0 - \vec{r}_{\text{cam}}\right\|}$$

ce qui nous donne un rayon

$$\vec{r}_{\text{ray}} = \vec{r}_0 + \vec{v}t$$

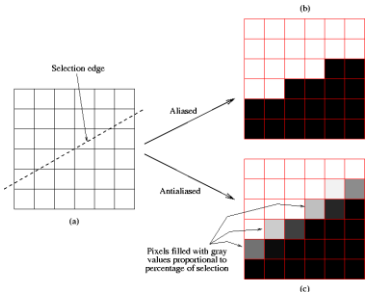
particulier pour tous les pixels de coordonnée (N_x , N_y) de l'écran.



Il ne reste plus qu'à tester l'intersection de ce rayon avec les géométries de la scène afin d'évaluer la couleur du rayon et ainsi colorer adéquatement le pixel (N_x , N_y).

Anticrénelage (*anti-aliasing*)

Le crénelage (*aliasing*) portant également le nom d'effet escalier survient lorsque l'on doit reproduire une courbe à couleur unique dans un espace en forme de grille régulière (écran de pixels). Pour réduire l'effet visuel, il faut calculer une couleur moyenne sur la ligne de démarcation des couleurs afin d'adoucir la transition des couleurs. Pour une ligne noire sur fond blanc, il faut remplacer certains pixels blanc et noir en pixels gris sous différentes teintes afin d'adoucir l'effet d'escalier.

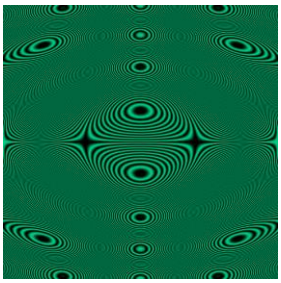


http://www.linuxtopia.org/online_books/graphics_tools/gimp_advanced_guide/gimp_guide_node36_007.html

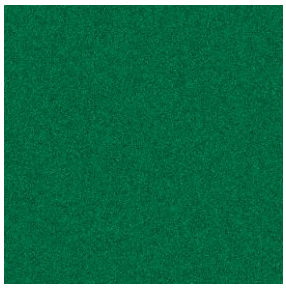
Pour réduire le crénelage (*anti aliasing*) dans un algorithme de *ray tracing*, il faut lancer plusieurs rayons dans chaque pixel avec des coordonnées internes différentes (pas toujours dans le coin supérieur gauche) et faire une moyenne des couleurs calculées. Ceci permet d'avoir une couleur « pondérée » lorsqu'un rayon touche la bordure d'une géométrie. Certains rayons d'un même pixel toucheront une géométrie et d'autres non ce qui donnera une couleur nuancée.

Cependant, ceci peut représenter une perte d'information si deux couleurs vraiment distinctes peuvent se retrouver dans le même pixel.

Exemple : Figure d'interférence



Patron d'interférence de Young (d = 2 mm, λ = 500 nm, L = 3m).



Disparition du patron d'interférence avec 5 rayons aléatoires.