Sin waves into water

fortement inspiré des vidéos d'Acerola sur YouTube.

1. Introduction

1.1. First approach

$\text{frequency} = w = \frac{2}{\text{wavelength}}$
$f(x) = \alpha * \sin(x + w)$

1.2. Phase

$\varphi = \text{speed} * \frac{2}{\text{wavelength}}$
$f(x,t) = \alpha * \sin(x * w + t * \varphi)$

1.2.1. étape 1

afficher une somme de sin en 2D

1.2.2. étape 2

afficher une somme de sin en 3D
Un plan composé de pleins de sommets.
$\text{vertex count} = (\text{xSize} + 1) * (\text{zSize} + 1)$
Il faut déplacer chaque sommets en fonction de la somme de sin.
Il faudrait itérer sur chacun des sommets et calculer sa nouvelle position à chaque étape.
Pour faire ça -> il faut mieux utiliser le GPU, qui affichera dans l'étape de la chaine de rendu qui est :
The vertex Shader
Il va calculer chaque positions de sommets en fonction de la somme de sin.

2. Pixel shader

il faut regarder comment la lumière interagit avec chaque point de la surface de l'eau.
utilisation du principe de **Lambertian diffuse**
Il y a une relation entre la direction de la source de lumière et la direction de la surface, ce qui est la **surface normal**. (un vecteur orthogonal à la surface (angle droit)). Plus l'angle entre le vecteur de la source lumineuse et le vecteur normal en un point de la surface est grand, moins la lumière est dispersée sur la surface.
La fonction cos(x) donnera 1 si l'angle = 0° et −1 si l'angle = 180°. Toutes les valeurs en dessous de 0 de la fonction cos(x) sont ammenées à 0 (la lumière négative n'existe pas).

2.1. Comment avoir l'angle ?

Dot product : $\vec{N} \cdot \vec{L} = \left|\vec{N}\right|\left|\vec{L}\right| \cos(\theta)$
Vecteurs normaux (N) et vecteur lumineux(L) sont normalisées !
donc $\vec{N} \cdot \vec{L} = \cos(\theta)$
$\vec{N} \cdot \vec{L} = \vec{N}_x * \vec{L}_x + \vec{N}_y * \vec{L}_y + \vec{N}_z * \vec{L}_z = \text{value between 0 and 1.}$ Description de combien la valeur de la surface doit être brillante

2.2. Comment avoir le vecteur normal ?

La surface normale : un vecteur qui est perpendiculaire à la surface à n'importe quel point donné.
Pour avoir ces vecteurs il faut 2 autres vecteurs :
Tangent et **Binormal**

Tangent : la line tangeante sur l'axe des X

Binormal : la line tangeante sur l'axe des Z
Le produit vectoriel de ces vecteurs produit le vecteur normal

Pour avoir le vecteur Tangent et Binormal il faut calculer la pente dans l'axe des X et des Y. On peut l'estimer avec la formule de la pente slope $= \frac{\sin(x+1)-\sin(x-1)}{(x+1)-(x-1)}$. Mais méthode couteuse et approximative.
On peut aussi la calculer avec la méthode des **dérivé partielle**.
$\frac{\partial}{\partial x} = \vec{d} * \cos(d \cdot (x, z))$ dérivée partielle en x

$\frac{\partial}{\partial z} = \vec{d} * \cos(d \cdot (x, z))$ dérivé partielle en z

$\vec{T} =< 1, 0, \frac{\partial}{\partial x} >$ vecteur Tangente
$\vec{B} =< 0, 1, \frac{\partial}{\partial z} >$ vecteur binomial

$\vec{N} = \vec{T} \times \vec{B}$ vecteur normal !

Bon pour un sin, mais qu'en est-il de la somme de sin ?

2.3. sommes de sin

somme de sin : $\sum_0^n \alpha_i * \sin(x * w_i + t + \varphi_i)$.

comme $\frac{\partial}{\partial x} \sum_0^n \Leftrightarrow \sum_0^n \frac{\partial}{\partial x}$, on peut (dans la même boucle pour plus d'efficacité) calculer la somme de sin puis la somme des dérivés qui est :

$$\frac{\partial}{\partial x} \sum_0^n w_i * \alpha_i * \vec{d_x} * \cos(x * w_i + t * \varphi_i)$$

pour que l'eau paraisse bleu, il suffit de multiplier le diffuse par du bleu

2.4. specular reflection

$\vec{V}$: view vecteur (normalisé)
$\vec{L}$: Light vecteur (normalisé)
$\vec{H} = \vec{V} + \vec{L}$: Halfway vecteur
specular $= \left(\vec{H} \cdot \vec{N}\right)^{\text{size}}$ aussi appelé bling phong specular (pas très réaliste cependant)
DONC
lighting $=$ diffuse $+$ specular $+$ ambient

3. Plus de réalisme

3.1. types d'ondes

$\sum_0^n \sin()$ est en fait $\sum_0^n$ toutes les ondes que tu veux
Pour plus de réalisme et de vagues pointues on peut se servir d'autres fonctions comme :

$$e^{\sin(x)-1}$$

Nouvelle dérivée : $\frac{\partial}{\partial x} = \sum_0^n w_i * \vec{d_x} * \alpha_i * e^{\sin(x*w_i+t*\varphi_i)-1} * \cos(x * w_i + t * \varphi_i)$ Par contre, pour plus de réalisme, il faut plus de vagues. Comme ça fait n'importe quoi, il faut se servir du fractional brownian motion.

3.2. fractional brownian motion

brownian motion : mouvement complètement random
fractional brownian motion : mouvement random mais avec une mémoire
$\alpha_i = 1$
$w_i = 1$

$\vec{d} =$ random mouvement
$\alpha_i * \text{wave}\big(\vec{d} \cdot (x, z) * w_i + t * \varphi_i\big)$
idée :
$\alpha_i = 0.82^x$
$w_i = 1.18^x$
Plus la fréquence est élevée, moins l'est l'amplitude.

3.3. domain warping

notre somme de vagues : $\sum_0^n \alpha_i * \text{wave}(x * w_i + t + \varphi_i)$.
On peut augmenter x par la dérivé de l'ancienne vague : effet sympa de vagues

4. Génerer beaucoup d'ondes et eviter la périodicité

principe de Wave cascading, set de 4 sommes par exemple, pour éviter la périodicité, car les vagues vont s'obscurcir l'unes l'autres.
Sinon, balancer plus de vagues, grace à la transformé de fourier

4.1. domaine fréquenciel

notation : spectre = $\hat{h}(x)$

Concept de "fully developed sea" : une mer avec des vents lui soufflant dessus sur des centaines de km pendant plusieurs jours
Description d'une fully developed sea : **JONSWAP METHOD**
cela permet de contrôler :

- Wind speed
- Wind direction
- Wind fetch (distance from the closest shore)
- Direction distribution (combien de vagues dans le meme sens)
- Amplitude override (?)
- Low pass filter (remove high freq waves)

$P(k)$: JONSWAP spectrum function
Ou
$P(k) = \frac{e^{-\frac{1}{(kL)^2}}}{k^4}$
avec $L = \frac{V^2}{g}$
V c'est la windspeed et g la gravité (9.81)

ξ_r, ξ_i : gaussian distributed random numbers
$\hat{h}_0(k) = \frac{1}{\sqrt{2}}(\xi_r + i\xi_i)\sqrt{P(k)}$ -> do this in every pixels of the pictures (spectrum)
Une fois fait, appliquer la **transformée de fourier inverse**.

4.2. transformée de fourier inverse classique

faire la somme suivante pour tout les pixels de la carte de hauteur :
$h(x) = \sum_k \hat{h}(k)e^{ik\cdot x}$ nouvelle manière de calculer les dérivées partielles (apparement c'est plus beau) :
$\nabla h(x) = \sum_k ik * \hat{h}(k)e^{i}(k \cdot x)$

nouvelle manière de calculer le champs de vecteurs normaux (moins beau mais plus efficace) :
slope $= \frac{h(x+1)-h(x-1)}{(x+1)-(x-1)}$

4.3. phase
$\hat{h}(k,t) = \hat{h}_0(k) * e^{ift}$
$\hat{h}_0(k)$: spectre initial qui est stocké dans une texture, il n'a besoin d'être calculé qu'une seul fois.
$e^{it} = \cos(t) + i\sin(t)$
$f = \sqrt{|k|g}$: dispersion relation
$g = 9.81$
Mais c'est super lent

4.4. fast fourier transform

4.5. Vecteur horizontal pour plus de réalisme
$\vec{D}(x,t) = \sum_k -i\frac{k}{|k|}\hat{h}(k,t) * e^{ik\cdot x}$ transformée de fourier classique (la faire en rapide)
l'ajouter aux sommets pour des belles vagues :)

5. rendu

5.1. optimisation
gpu dynamic tesselation (voir moins les sommets qui sont loin de la caméra)

5.2. lighting model
Modèle actuelle : éclairage : ambient + diffuse + specular + env. reflection (reflection du ciel, elles font bcp de taff)
Ce modèle est bien si le matériel est opaque, ce qui n'est pas le cas de l'eau.
Nouveau modèle : ambient + scatter + specular + env. reflection
scatter formule :

$$L_{scatter} = (k_1 H\langle \omega_i \cdot -\omega_o\rangle^4 (0.5 - 0.5(\omega_i \cdot \omega_n))^3 + k_2\langle \omega_o \cdot \omega_n\rangle^2) C_{ss} L_{sun} \cdot \frac{1}{(1+\Lambda(\omega_i))}$$

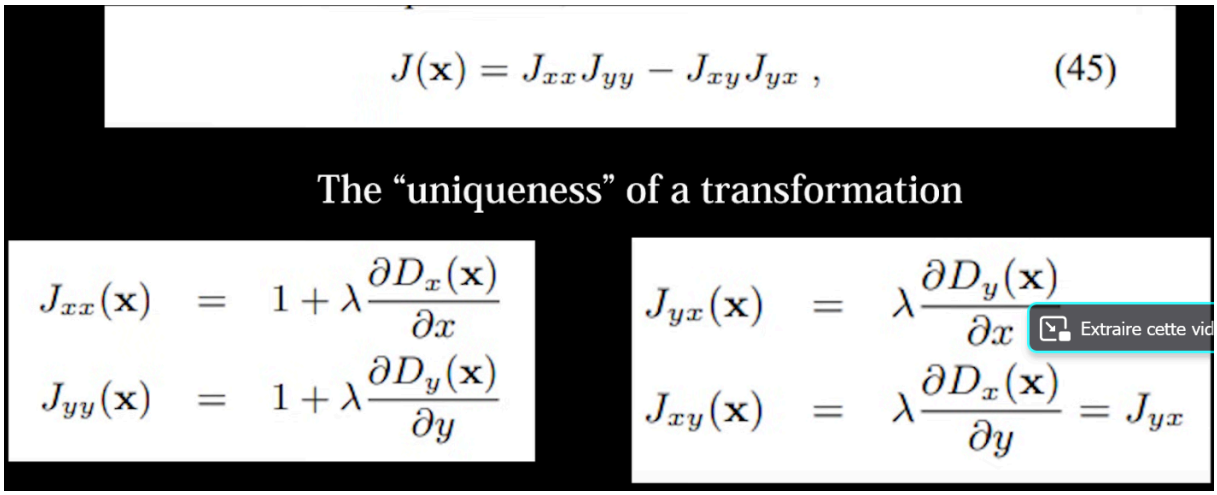
$$L_{scatter} \mathrel{+}= k_3\langle \omega_i \cdot w_n\rangle C_{ss} L_{sun} + k_4 P_f C_f L_{sun}$$

bleu et vert : dépendent de la hauteur des vagues.
rouge : à quel point la normal est visible à la caméra
violet : lambert's cosine law
jaune : ambient light value
Scatter color : bleu (l'eau est bleue)

Maintenant on améliore le specular qui était créé avec la formule de blinn phong par la méthode physically based rendering

5.3. écume !!!
différence entre **sea foam** (l'écume sur l'eau) et **sea spray** (particules d'écumes qui se font éjecter).

sea foam :
Elle apparait avec une vague qui se casse. Comment détecter si une vagues est cassée ? quand elle se retourne sur elle même. Les maths pour détecter ça :
Jacobian formule :



Ce qu’elle fait : elle détecte quand les vagues se retournent sur elles même et il faut ensuite générer de la mousse. Si on veut plus d’écume, il faut baisser la formule de Jacobian.
Attention, pas très réaliste, il faut aussi accumuler de l’écume pour un meilleur rendu.
Il faut donc suivre l’évolution de la génération d’écume en la stockant dans une texture. Quand le jacobien est négatif on ajoute un peu d’écume dans la texture. Puis à chaque frame, on applique une exponential decay sur la texture pour enlever la foam

5.4. éviter le tiling :O

Lancer 4 simulations de différentes tailles, et les mettre l’unes sur l’autres en empilant les tiles.

5.5. détails

- mettre des couleurs sympa pour l’ambient, diffuse et specular light
- mettre un flou sur l’horizon (distant fog post process), puis le réduire en fonction de la hauteur pour faire un effet atmosphere
- mettre un soleil sun $= \vec{V} \cdot \vec{L}$ et le blender avec la source, puis ajouter un bloom pass, puis l’adoucir en utilisant un cinematic tone maper.
- mettre un ciel (skybox (allsky free)) et le faire bouger pour faire genre les nuages bougent

5.6. réflection du ciel

3 options :

- Ray tracing (trop couteux)
- Screen space reflections (SSR) (trop moche)
- Image based reflections

On va utiliser **Image based reflections**
$\vec{R} = 2\vec{N}\left(\vec{N} \cdot \vec{V}\right) - \vec{V}$: reflected vecteur
Ajout de **Fresnel** : relation entre la force de la reflection et l’angle d’incidence
Calcul possible : fresnel $= \left(1 - \vec{V} \cdot \vec{N}\right)^5$
plus le vecteur caméra est proche du vecteur normal, moins la surface relfete la lumière.
Donc nouvelle formule du specular : specular $= \left(\vec{H} \cdot \vec{N}\right)^{\text{size}} *$ fresnel