جامعة دمشق
كلية الهندسة المعلوماتية
السنة الرابعة
الحقائق الافتراضية

محاكاة تسونامي

تقديم الطلاب:

رغد الحلبي

سدره ميرخان

عبد العليم السيد

عمر القبطان

محمد كريم الكل

الاثنين 13/6/2022

جدول المحتويات:

خطوات المحاكاة:

تم الاعتماد على خوارزمية SPH، بحيث يتم حساب الكثافة والضغط معاً بدايةً، ومن ثم يتم حساب القوى الداخلية (الضغط واللزوجة)، والخارجية (الجاذبية والتوتر السطحي)، أخيرا يتم تطبيق الـ integration وتحديد التصادم.

حساب الكتلة والكثافة والضغط:

الكتلة والكثافة:

لتطبيق قوانين القوى المستخدمة في SPH نحتاج لمعرفة كتلة وكثافة الـparticle.

إنّ كتلة الـparticle يتم إدخالها من قبل المستخدم وتبقى ثابتة، أما الكثافة فيتم حسابها وفق القانون التالي:

$$\rho_i = -\rho(\vec{r}_i) = -\sum_{j}^{neighbors} m_j W_{polynomial}(\vec{r}_i - \vec{r}_j, h)$$

حيث $W_{polynomial}$ هو الـ kernel المستخدم في حساب الكثافة.

الضغط:

ويحسب اعتماداً على الكثافة عن طريق Tait equation:

$$p = P_0 \left[\left(\frac{\rho}{\rho_0} \right)^{\gamma} - 1 \right]$$

حيث:

$\rho_0 = 1000\ kg/m^3$ ويمثل الكثافة المرجعية $restDensity$.

$\gamma = 7$ وهي القيمة التي تستخدم مع المياه.

$P_0 = const$

حساب القوى:

القوى الداخلية:

القوة الناتجة عن الضغط:

تعطى قوة الضغط من أجل كل particle (i) بالعلاقة الآتية:

$$f_i^{pressure} = -\nabla p(r_i) = -\sum_{j \neq i}^{neighbors} \frac{p_i + p_j}{2} \frac{m_j}{\rho_j} \nabla W_{spiky}(\vec{r_i} - \vec{r_j}, h)$$

حيث:

- m_j تمثّل كتلة الparticle المجاور j.
- p تمثّل الضغط الذي تم حسابه سابقا.
- $W_{spiky}(\vec{r_i} - \vec{r_j}, h)$ يمثل الKernel/weight function.

قوة اللزوجة:

عند تدفق السائل، تخضع جزيئاته لاحتكاك داخلي والذي يقلل الطاقة الحركية للجزيئات محولاً إياها إلى حرارة.

إنّ مقاومة التدفق تدعى باللزوجة (*viscosity*) ومعامل اللزوجة μ يعرف مدى لزوجة السائل.

تعطى قوة اللزوجة من أجل كل particle (i) بالعلاقة الآتية:

$$f_i^{viscous} = \mu \nabla^2 \vec{v}(\vec{r_i}) = \mu \sum_{j \neq i}^{neighbors} (\vec{v_j} - \vec{v_i}) \frac{m_j}{p_j} \nabla^2 W_{visc}(\vec{r_i} - \vec{r_j}, h)$$

حيث:

m_j تمثّل كتلة الparticle المجاور j

$\vec{v_j}$ تمثّل سرعة الparticle المجاور j

μ تمثّل معامل اللزوجة

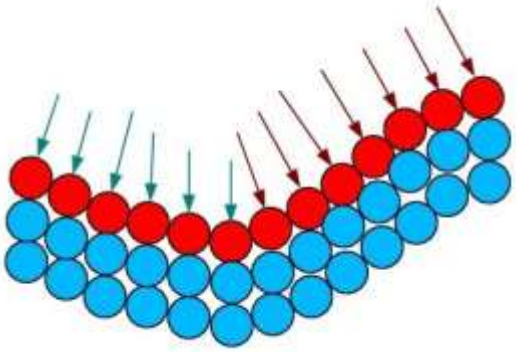
$W_{visc}(\vec{r_i} - \vec{r_j}, h)$ يمثل الKernel/weight function

القوى الخارجية:

قوة الجاذبية:

إنّ قوة الجاذبية تؤثر بشكل متساوي على كل جزيئات السائل، وهي تعبر عن تسارع الجاذبية الأرضية لذلك يتم جمعها إلى قيمة التسارع عند القيام بعملية *Integrate*.

قوة التوتر السطحي:



الجسيمات الموجودة على سطح السائل تخضع لقوى غير متوازنة ، وهذا ينتج عنه التوتر السطحي. ويكون اتجاه هذه القوة دائماً باتجاه السطح الداخلي للسائل.

سطح السائل يمكن أن يعطى باستخدام color field، ويعبر عنه بالقانون:

$$C_s = \sum_{j}^{neighbors} \frac{m_j}{\rho_j} W_{polynomial}(\vec{r_i} - \vec{r_j}, h)$$

مشتق color field : $n = \nabla c_s$

قوة التوتر السطحي من أجل كل particle i:

$$f_i^{surface} = -\sigma \nabla^2 c_s \frac{n}{|n|}$$

ويتم حساب التسارع من القانون التالي:

$$a = \frac{f}{\rho}$$

توابع الـ kernels المستخدمة:

1) $W_{polynomial}(\vec{r},h) =$

$$\begin{cases} \frac{315}{64\pi h^9}(h^2 - \|r\|^2)^3 & 0 \le \|\vec{r}\| \le h \\ 0 & \|\vec{r}\| > h \end{cases}$$

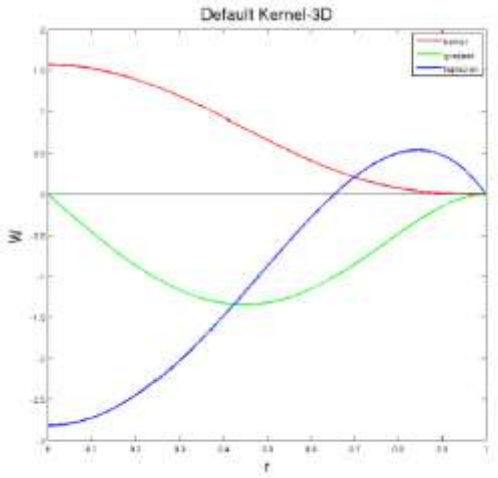


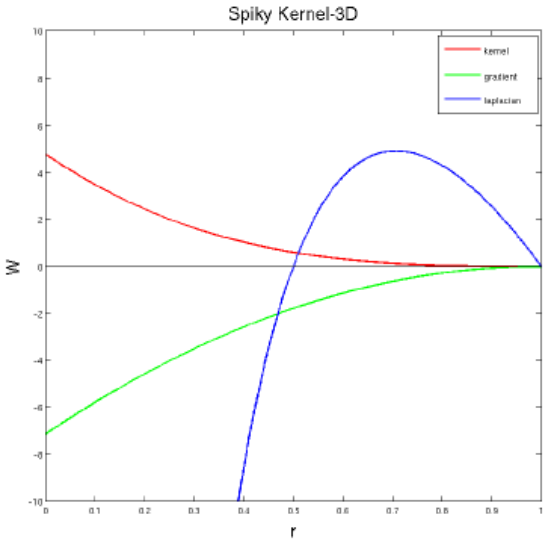
FIGURE 2.3: Polynomial Kernel, its Gradient and Laplacian

ومن W نحسب الـ *gradient* و *Laplacian* كالآتي:

$$\nabla W_{poly}(\vec{r},h) = -\frac{945}{32\pi h^9}.\vec{r}\,(h^2 - \|r\|^2)^2$$

$$\nabla^2 W_{poly}(\vec{r},h) = -\frac{945}{32\pi h^9}.(h^2 - \|r\|^2)(3h^2 - 7\|r\|^2)$$

2) $W_{spiky}(\vec{r},h) = \begin{cases} \frac{15}{\pi h^6}(h - \|r\|^3) & 0 \le \|\vec{r}\| \le h \\ 0 & \|\vec{r}\| > h \end{cases}$



حيث:

h تمثل smoothing length وتحدد نص قطر مجال التأثير Ω

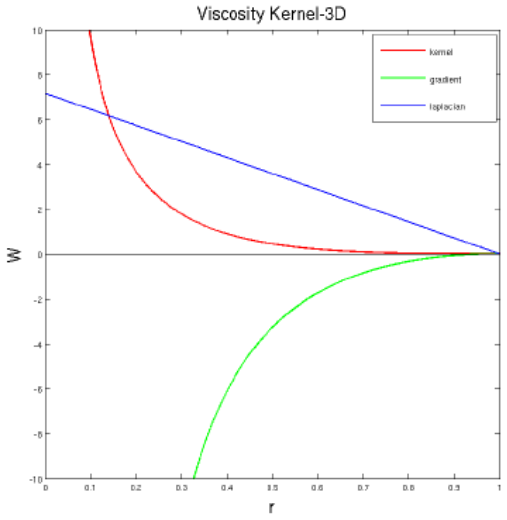
$$\vec{r} = \vec{r_i} - \vec{r_j}$$

ومن W نحسب الـ *gradient* كالآتي:

$$\nabla W_{spiky}(\vec{r},h) = -\frac{45}{\pi h^6}.\frac{\vec{r}}{\|\vec{r}\|}(h - \|r\|)^2$$

3) $W_{visc}(\vec{r},h) =$

$$\frac{15}{2\pi h^3}\begin{cases} -\frac{\|r\|^3}{2h^3} + \frac{\|r\|^2}{h^2} + \frac{h}{2\|r\|} - 1 & 0 \le \|\vec{r}\| \le h \\ 0 & \|\vec{r}\| > h \end{cases}$$



ومن W نحسب الـ*gradient* و *Laplacian* كالآتي:

$$\nabla W_{visc}(\vec{r},h) = -\frac{15}{2\pi h^3}.\vec{r}\left(-\frac{3\|\vec{r}\|}{2h^3} + \frac{2}{h^2} - \frac{h}{2\|\vec{r}\|^3}\right)$$

$$\nabla^2 W_{visc}(\vec{r},h) = -\frac{45}{2\pi h^6}.(h - \|r\|)$$

الصدم:

Wall boundary:

تم استخدام طريقة تسمى ب plenty method حيث يؤخذ الجداء الداخلي بين شعاع الوجه الذي سيتم تطبيق الصدم عليه مثلا الوجه الموازي لمحور x و الذي موضعه 10 يكون ناتج الجداء (1,0,0,10) * dist = (position.x, 0,0,1) هو البعد بين موقع البار تكل على المحور x و الوجه فإذا كانت القيمة سالبة هذا يعني ان البارتكل قد اصطدم أو فعليا تجاوز الوجه و من ثم نجمع لتسارع البارتكل (1,0,0) * -wallstifness * min(dist,0) حيث ال wallstifness هو معامل الصدم و الذي يعبر فعليا عن سماحية تجاوز البارتكل للجدار و الشعاع(1,0,0) لكي يصبح المعامل wallstifness شعاع بجهة ال x فقط , فإذا كانت قيمة min(dist,0) هي صفر هذا يعني أن البارتكل لم يصل للجدار و لا يوجد صدم و سيتم جمع صفر للتسارع.

ويتم تكرار هذه العملية على جميع الأوجه المراد الصدم بها.

Buildings:

بنفس الطريقة السابقة ولكن بدون جداء داخلي لمعرفة المسافة بين البارتكل والجدار، إنما تم استخدام شروط بسيطة حيث تم حساب المسافة بين البارتكل والجدر الأربعة للبناء ويتم صدم البارتكل مع الجدار الأقرب مسافةً بشرط أن يكون البارتكل قد لامس البناء.

طرائق الـ integration:

لحساب السرعة والموضع الجديدين بكل لحظة زمنية لكل particle i استخدمنا طريقة أويلر:

$$v_i^{k+1} = v_{iold}^k + a_i^k * dt \qquad r_i^{k+1} = r_i^k + v_i^{k+1} * dt$$

تحسين الأداء:

Rendering:

1) قمنا برسم الجزيئات دون استخدام الـ built-in textures حيث قمنا برسمها كدوائر 2D لتحسين المعالجة، ومن خلال زاوية الـ camera تبدو كما لو كانت 3D، كما أنه يتم من خلال الـ buffer ذاتها التي يتم فيها تخزين معلومات الجزيئات من قبل الـ gpu مباشرة من أجل عملية الرسم دون التخزين على الـ RAM.
2) لجعل المحاكاة تبدو واقعية بحيث تظهر الجزيئات كسائل متماسك قمنا باستخدام طريقة Voronoi لحساب لون الجزيئات حيث يتم إضافة قيمة noise معينة تبعاً لأقرب نقطة مجاورة وقيمة عشوائية (وبما أن من الصعب تطبيق تابع random في shader تم الاعتماد على unity_DeltaTime) وهكذا يظهر كل جزيء بلون معين اعتمادا على نمط أقرب مجاور له ويعطي هذا تأثيرا قريبا للماء.

Shaders:

إن استخدام الـ shaders يتيح الاستفادة من قوة المعالجة لوحدة معالجة بطاقة الرسومات (GPU) بدلاً من الاعتماد فقط على وحدة المعالجة المركزية للنظام (CPU)، ولهذا قمنا بتنفيذ جميع الحسابات والرسم أيضا على الـ GPU.

كما أن المعالجة من خلال shaders تكون على التوازي باستخدام threads، وهكذا يتم تنفيذ العمليات الحسابية لكل جزيء على التوازي وبشكل مستقل عن الجزيئات الأخرى.

Spatial Hashing:

أحد أهم أجزاء المحاكاة المهمة و التي تستغرق الكثير من الوقت هو البحث عن المجاورات من أجل كل جزيء، بداية كنا قد استخدمنا الطريقة البدهية للبحث وهي أنه بالنسبة لكل جسيم نقوم بالمرور على جميع الجزيئات وذلك للعثور على تلك الموجودة ضمن نصف قطر معين (h_smoothlen).

إن تعقيد هذه الطريقة هو $O(n^2)$ حيث n هو عدد الجزيئات.

عندما يكون عدد الجزيئات كبيرا، تكون هذه الخوارزمية بطيئة جدا ولذلك قمنا بتطبيق خوارزمية التجزئة المكانية أو الـ Spatial hashing وفيها، يتم الاعتماد على حقيقة أنه لكل جسيم i نصف قطر هو الـ smoothing radius، وببساطة فإن الجزيئات التي تقع على مسافة $h <$ من الجزيء i ليس لها أي تأثير عليه.

يتم ذلك عن طريق التقسيم إلى خلايا بحجم 2h، وهكذا يصبح لدينا عدد صغير من الخلايا التي يجب البحث ضمنها لضمان العثور على جميع الجيران المحتملين.

تنقسم الخوارزمية بشكل أساسي إلى خطوتين أساسيتين:

الأولى: هي إدخال الجزيئات عن طريق الـ hashing ووضعها في خلاياها الصحيحة.

الثانية: هي الاستعلام عن الخلايا المناسبة لمجاورات جسيم معين.

تفرض أن جميع الجسيمات على مسافة> h بعيدة عن الجسيم وليس لها أي تأثير عليه. من خلال تقسيم العالم إلى خلايا بحجم h في كل اتجاه ، لا يوجد سوى عدد صغير من الخلايا التي يجب البحث عنها لضمان العثور على جميع الجيران المحتملين.

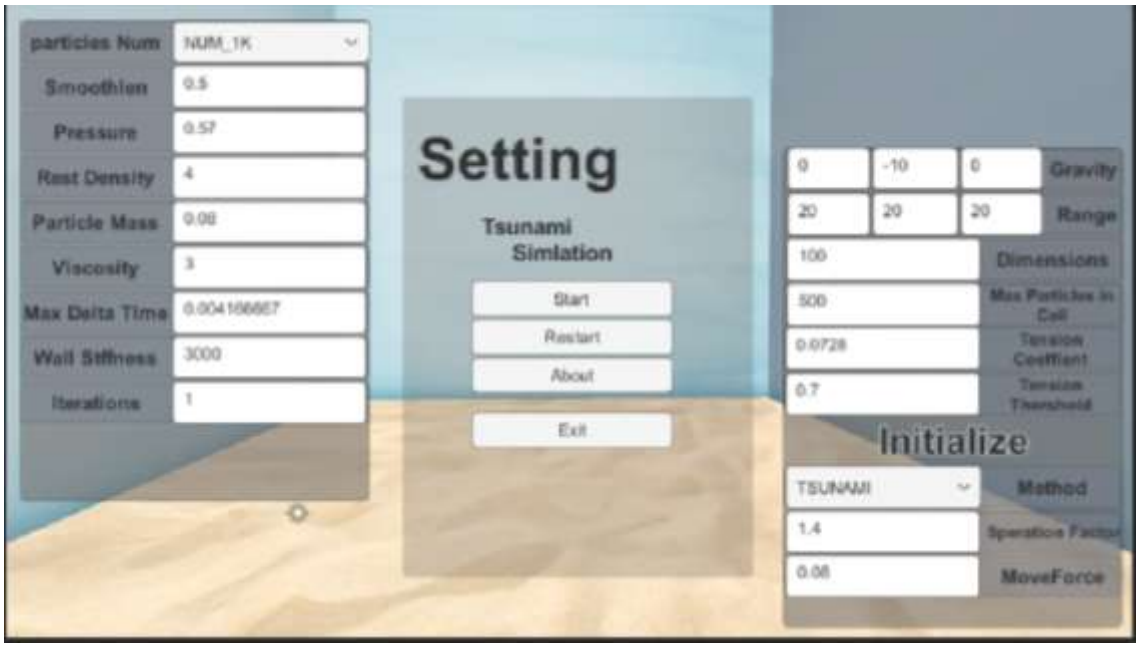
لهذه الخوارزمية تعقيد زمني $O\ (mn)$، حيث m هو متوسط عدد الجزيئات الموجودة و n هو عدد الجزيئات.

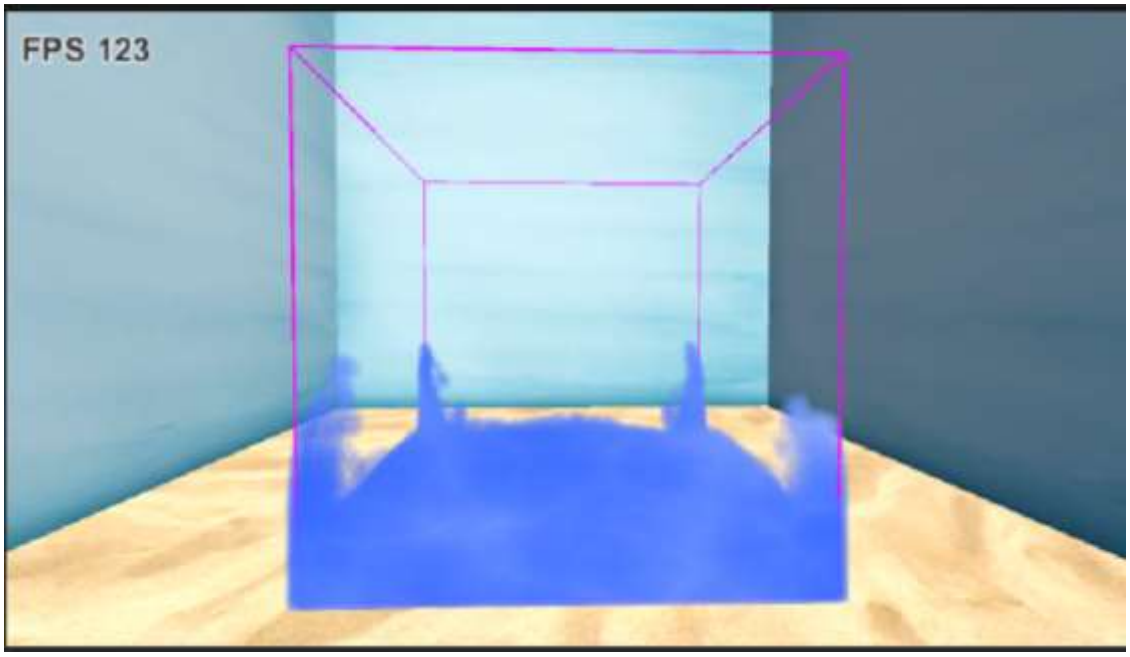
الثوابت و الـ parameters:

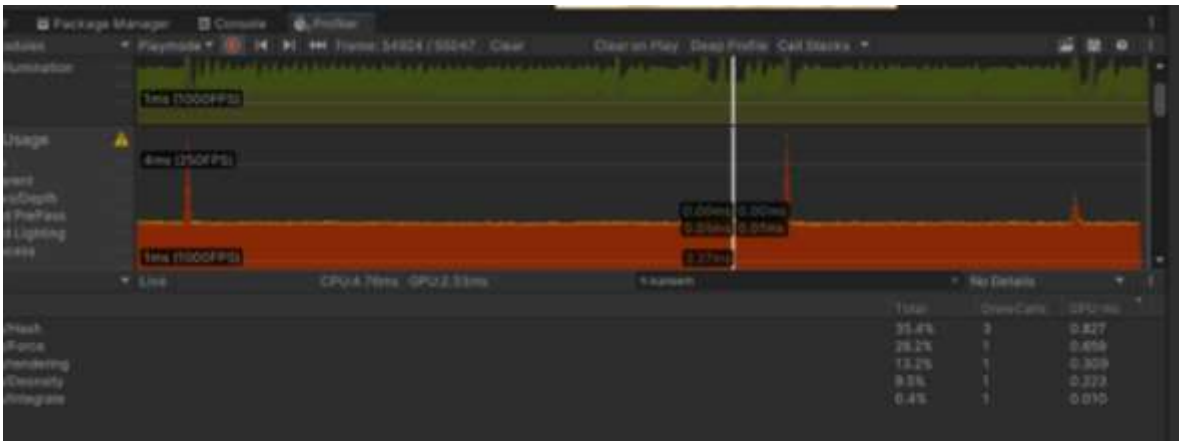
```
int   _NumParticles;      // Number of particles
float _TimeStep;          // Time step width (dt)
float _Smoothlen;         // Particle radius
float _PressureStiffness; // Becker's coefficient
float _RestDensity;       // resting density
float _DensityCoef;       // Coefficient when calculating density
float _GradPressureCoef;  // Coefficient when calculating pressure
float _LapViscosityCoef;  // Coefficient when calculating viscosity
float _GradTensionCoef;   // Coefficient when calculating Tension
float _LapTensionCoef;    // Coefficient when calculating Tension
float _WallStiffness;     // The pushing force of the penalty method
float _Viscosity;         // Viscosity coefficient
float3 _Gravity;          // gravity
float3 _Range;            // Simulation space
float _tensionThreshold;  // Tension Threshold
float _tensionCoefficient; // Tension Coefficient
float _Damping;           // Damping for wall boundary
float3 _Origin;           // reference point

float3 _MousePos;         // Mouse position
float _MouseRadius;       // Radius of mouse interaction
bool _MouseDown;          // Is the mouse pressed
float CellSize;
int Dimensions;
int maximumParticlesPerCell;
float particleRadius;
```

النتائج:

واجهة الإدخال:







عدد الجزيئات الذي تم تحقيقه: 256 آلف جزيء.

السرعة fps: 40.

المشاكل التي تمت مواجهتها:

- لقد كانت بيئة Unity محدودة من أجل توسيع المشروع أكثر مما كانت مفيدة، فعلى الرغم من سهولة بناء الـ scenes من خلالها الذي جعل عملية الاختبار أكثر سلاسة والقدرة على تعديل الـ parameters المختلفة من خلال الـ inspector إلا أن معظم العوائق خلال عملية تطوير المشروع كانت في محاولات تحسين الأداء وباستخدام بيئة Unity لا يمكن سوى استخدام لغة C# والتي تعد بطيئة نسبة إلى لغات أخرى مناسبة أكثر من أجل عملية تطوير engines مثل لغات C++ أو Rust أو Fortran.

- مشاكل التعامل مع shaders:

على الرغم من أن الـ shaders أكسبت المشروع قدرة عالية لمعالجة عدد كبير من الجزيئات من خلال تسريع الحسابات إلا أنها جعلت من عمليات كتابة clean code واختباره والـ debug أصعب نتيجة محدودية التعامل معها من نواحي عدة مثل صعوبات التعامل مع المصفوفات وتواصل الـ GPU مع الـ CPU من أجل عمليات الطباعة وغيرها.

رابط المشروع: https://github.com/kareemalkoul/Tsunami

المراجع المستخدمة:

https://journals.tdl.org/icce/index.php/icce/article/view/10377

https://www.diva-portal.org/smash/get/diva2:573583/FULLTEXT01.pdf

https://www.researchgate.net/publication/278085555_SPH_modeling_of_tsunami_waves

https://www.cs.cornell.edu/~bindel/class/cs5220-f11/code/sph.pdf

https://www.dive-solutions.de/articles/sph-basics

Smoothed Particle Hydrodynamics: A Meshfree Particle Method: Liu, G. R., Liu, M. B., Liu, GUI-Rong: 9789812384560: Amazon.com: Books

Start here! (article) | Effects | Khan Academy

https://link.springer.com/content/pdf/10.1007%2F3-540-47789-6_11.pdf

https://github.com/Gornhoth/Unity-Smoothed-Particle-Hydrodynamics/tree/master/MonoBehaviour_ComputeShader
https://iquilezles.org/articles/voronoilines/
https://www.shadertoy.com/view/4tXSDf