Final Report by Kevin Pick

Implementation of "A material point method for snow simulation" from UCLA and Walt Disney Animation Studios

**Overview**

The goal of the implementation was to recreate the Disney snow paper on the GPU as it was originally implemented on the CPU. The program is run on an RTX 2080 with 8GB of VRAM utilizing CUDA to parallelize the program. Since each stage presented in the algorithm section requires independent computations or dependencies that can be resolved with atomic addition, the conversion of the algorithm to utilize the GPU is trivial. The only problematic computations are the polar decomposition and SVD. For both kernels, preexisting CUDA implementations were used.

**Basic Concept**

As stated in class, the deformation of a body can be described as a mapping from the undeformed configuration ***X*** to a deformed configuration ***x*** as follows: $x = \varphi(X) = F\,X + c$. Hence, $F$ must be a 3 x 3 matrix deformation gradient, $F = \frac{\partial \varphi}{\partial X}$, by the aforementioned equation. Since the deformation must obey the laws of physics, mass and momentum are conserved and the body must change according to an elasto-plastic constitutive relation: $\frac{Dp}{dt} = 0, \rho\frac{Dv}{dt} = \nabla \cdot \sigma + \rho g, \sigma = \frac{1}{J}\frac{\partial \Psi}{\partial F_E}F_E^T$. Please note, $\rho$ is density, $t$ is time, $v$ is velocity, σ is the Cauchy stress, $g$ is gravity, $\Psi$ is the energy density of the elasto-plastic potential, $F_E$ is the elastic portion of the deformation gradient $F$, and $J = \det(F)$.

$$N(x) = \begin{cases} \frac{1}{2}|x|^3 - |x|^2 + \frac{2}{3}, & 0 \le |x| < 1 \\ \frac{-1}{6}|x|^3 + |x|^2 - 2|x| + \frac{4}{3}, & 1 \le |x| < 2 \\ 0, & otherwise \end{cases}$$

The snow is represented via particles which require the position $x_p$, velocity $v_p$, mass $m_p$, and deformation gradient $F_p$ to be stored. To compute $\rho\frac{Dv}{dt} = \nabla \cdot \sigma$, each particle is transferred to a regular grid via the 1D B-spline curve function $N(x)$.

$$N(d) = \begin{cases} \frac{1}{2}d^3 - d^2 + \frac{2}{3}, & 0 \le d < 1 \\ \frac{-1}{6}d^3 + d^2 - 2d + \frac{4}{3}, & 1 \le d < 2 \\ 0, & otherwise \end{cases} \quad N_d(d) = \begin{cases} \frac{3}{2}d^2 - 2d, & 0 \le d < 1 \\ \frac{-1}{2}d^2 + 2d - 2, & 1 \le d < 2 \\ 0, & otherwise \end{cases}$$

The function computes the weight contribution of each particle's field to a node's field on the regular grid. For a particle at position $x_p = \{x_x, x_y, x_z\}$ and a node located at $n_p = \{n_x, n_y, n_z\}$ on a grid with uniform spacing $h$, $w_{n_p} = N_n^h(x_p) = N\left(\frac{1}{h}(x_x - nh)\right)N\left(\frac{1}{h}(x_y - nh)\right)N\left(\frac{1}{h}(x_z - nh)\right)$. These weights are used to transfer momentum and mass from the particle to the grid. Hence, only particles within a distance of 2h of the node will contribute to its fields.

$$N_{n_w}^h(x_p) = \begin{cases} N_x\left(\frac{1}{h}(x_x - n_x h)\right) N\left(\frac{1}{h}(x_y - n_y h)\right) N\left(\frac{1}{h}(x_z - n_z h)\right) \\ N\left(\frac{1}{h}(x_x - n_x h)\right) N_y\left(\frac{1}{h}(x_y - n_y h)\right) N\left(\frac{1}{h}(x_z - n_z h)\right) \\ N\left(\frac{1}{h}(x_x - n_x h)\right) N\left(\frac{1}{h}(x_y - n_y h)\right) N_z\left(\frac{1}{h}(x_z - n_z h)\right) \end{cases}$$

Lastly, the gradient of $N(x)$ is required to update force. By defining distance variable $d = |x|$, we can transform $N(x)$ into $N(d)$ and then define the gradient as $N_x(x) = sign(x) * N_d(d)$ to avoid differentiating an absolute value function. Thus, $\nabla w_{n_p} = \nabla N_n^h(x_p) = \{N_{n_x}^h(x_p), N_{n_y}^h(x_p), N_{n_z}^h(x_p)\}$.

**Algorithm**

Assume we are on time step $t^\alpha$ and have the associated position, velocity, mass, and deformation gradient of each particle $p \in P$. Now, we attempt to advance the simulation by timestep $\Delta t$ to obtain $t^{\alpha+1}$.

1. Transfer particle fields to the grid using the weighting function:

   $$m_n^\alpha = \sum_{p \in P} m_p w_{n_p}^\alpha$$

   Transfer velocity while maintaining conservation of momentum by normalizing the resultant value by $m_n^\alpha$:

   $$v_n^\alpha = \frac{1}{m_n^\alpha} \sum_{p \in P} m_p w_{n_p}^\alpha v_p^\alpha$$

2. Calculate particle volume and density since force discretization requires volume in the initial configuration. We consider the density of a cell in the grid to be the cell's mass over $h^3$.

   $$\rho_p^0 = \frac{1}{h^3} \sum_n w_n^0 m_n^0$$

   We approximate the volume of a particle as follows:

   $$V_p^0 = \frac{m_n}{\rho_p^0}$$

3. To compute the forces within the grid:
   a. The deformation gradient $F = F_E F_P$ is a combination of the elastic $F_E$ and plastic $F_P$ values. In the base state, the elastic and plastic portions of the gradient are set to the identity matrix as the system is undeformed.
   b. Linear elasticity requires two Lamé parameters $\lambda$ and μ to be set. We utilize the suggested parameters from the paper for the initial Young's modulus, $E_0$, and the Possion's ratio, $v$:

      $$\lambda_0 = \frac{E_0 v}{(1+v)(1-2v)} \text{ and } \mu_0 = \frac{E_0}{2(1+v)}$$

   c. The constitutive model uses the elasto-plastic energy density function with the Lamé parameters defined as functions of the plastic deformation gradient.

$$\Psi(F_E, F_P) = \mu(F_P)||F_E - F_P||_F^2 + \frac{\lambda(F_P)}{2}(J_E - 1)^2$$

$$\mu(F_P) = \mu_0 e^{\xi(1-J_P)} \text{ and } \lambda(F_P) = \lambda_0 e^{\xi(1-J_P)}$$

$$J_E = \det(F_E) \text{ and } J_P = \det(F_P)$$

$R_E$ is the orthonormal component of the polar decomposition of $F_E = R_E S_E$

$\xi$ is the parameter for constant of plastic hardening

d. Node $n$ has the following force acting on it where the volume of the material of particle $p$ at time $\alpha$ with Cauchy stress $\sigma_p$ is $V_p^\alpha = J_p^\alpha V_p^0$:

$$force_n(x) = -\sum_{p \in P} V_p^\alpha \sigma_p \nabla w_{n_p}^\alpha$$

$$\sigma_p = \frac{1}{J_p^\alpha}\frac{\partial}{\partial F_E}\Psi(F_{E_p}^\alpha, F_{P_p}^\alpha)(F_{E_p}^\alpha)^T = \frac{2\mu\left(F_{P_p}^\alpha\right)}{J_p^\alpha}(F_{E_p}^\alpha - R_{P_p}^\alpha)(F_{E_p}^\alpha)^T + \frac{\lambda\left(F_{P_p}^\alpha\right)}{J_p^\alpha}(J_{E_p}^\alpha - 1)J_{E_p}^\alpha I$$

4. Grid velocities can be update according to standard formulas:

$$v_n^* = v_n^\alpha + \nabla t \frac{force_n^\alpha}{m_n^\alpha}$$

5. Collisions between grid cells and objects must be taken into account such that when a collision occurs, $\varphi(x_n) \leq 0$, the normal, $n = \nabla\varphi(x_n)$, and the colliding object's velocity, $v_{col}$, are computed:

Transfer the grid velocity to relative velocity in the collision frame of the colliding object:

$$v_{rel} = v_n^* - v_{col}$$

Normal component magnitude of the relative velocity:

$$v_n = \alpha * v_{rel}$$

If $0 \leq v_n$, meaning bodies are separating, no collision is applied; otherwise, the velocity in the tangent direction is computed:

$$v_t = v_{rel} - \alpha * v_\alpha$$

If the following is true ($\mu$ is the friction coefficient):

$||v_t|| \leq -\mu v_\alpha$, then $v'_{rel} = v_t + \frac{\mu v_\alpha}{||v_t||}v_t = (1 + \frac{\mu v_\alpha}{||v_t||})v_t$

Otherwise:

$$v'_{rel} = 0$$

Lastly, the velocity must be moved back into the world frame:

$$v^{*\prime}{}_n = v'_{rel} + v_{col}$$

6. Integration is currently performed explicitly $v_n^{\alpha+1} = v_n^*$.
7. The deformation gradient needs to be updated according to the following procedure:

a. Update the deformation gradient.

$$F_p^{\alpha+1} = (\nabla t v_p^{\alpha+1} + I) F_{E_p}^{\alpha} F_{P_p}^{\alpha}$$

$\hat{F}_{E_p}^{\alpha+1} = (\nabla t v_p^{\alpha+1} + I) F_{E_p}^{\alpha}$ and $\hat{F}_{P_p}^{\alpha+1} = F_{P_p}^{\alpha}$ for temporary changes to elastic portion of the deformation gradient

b. To add controls to the simulation, we define a critical deformation threshold which is applied as follows:

$$\hat{F}_{E_p}^{\alpha+1} = U_p \hat{\Sigma}_p \mathrm{V}^T{}_p \text{ (SVD computation)}$$

$$\Sigma_p = clamp(\hat{\Sigma}_p, [1 - \Theta_c, 1 + \Theta_s])$$

$\Theta_c$ is the critical compression ratio and $\Theta_s$ is the critical stretch ratio of the material

c. Using the clamped SVD computation of the elastic component of the deformation gradient, we can compute the plastic component of the deformation gradient by recombining the values.

$$F_{E_p}^{\alpha+1} = U_p \Sigma_p \mathrm{V}^T{}_p, F_{P_p}^{\alpha+1} = V_p {\Sigma_p}^{-1} U_p^T F_p^{\alpha+1}$$

d. The particle velocities must be updated according to the rules of a PIC (particle-in-cell) / FLIP (fluid-implicit-particle) simulation:

$v_p^{\alpha+1} = lerp(v_{PIC_p}^{\alpha+1}, v_{FLIP_p}^{\alpha+1}, \beta)$ where $\beta$ is some constant

$$v_{PIC_p}^{\alpha+1} = \sum_n v_n^{\alpha+1} w_{n_p}^{\alpha}$$

$$v_{FLIP_p}^{\alpha+1} = v_p^{\alpha} + \sum_n (v_n^{\alpha+1} - v_n^{\alpha}) w_{n_p}^{\alpha}$$

8. Particle body collision uses the same formulas as grid body collision where:

$$v_{rel} = v_p^{\alpha+1} - v_{col}$$

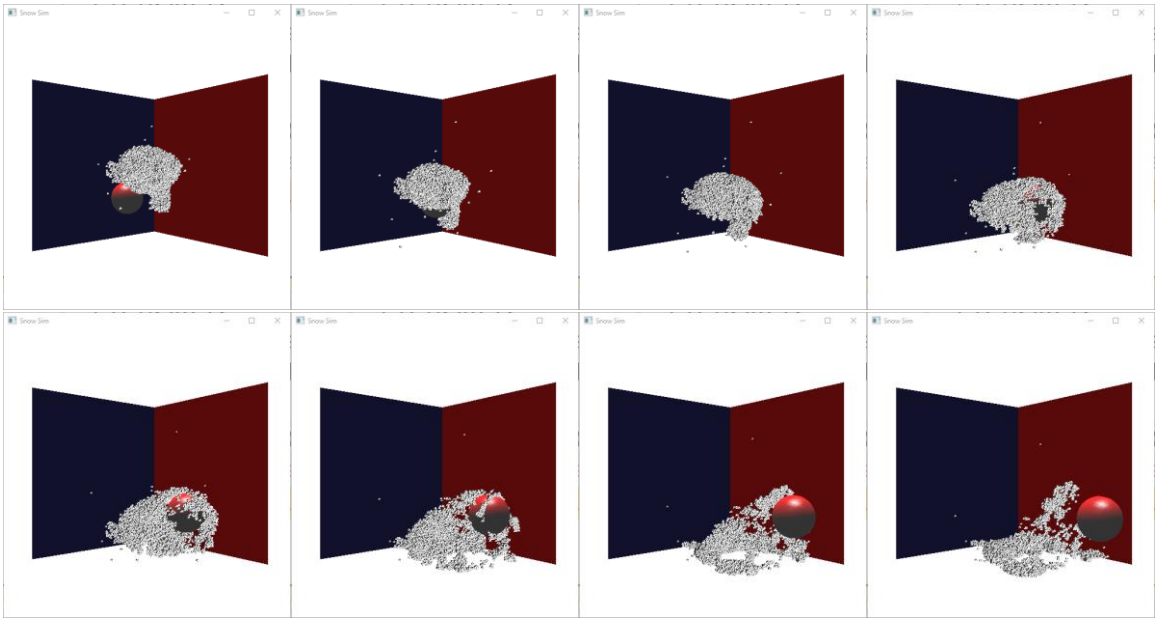
9. Lastly, we update the particle's position via a standard formula:

$$x_p^{\alpha+1} = x_p^{\alpha} + \nabla t v_p^{\alpha+1}$$

**Results**

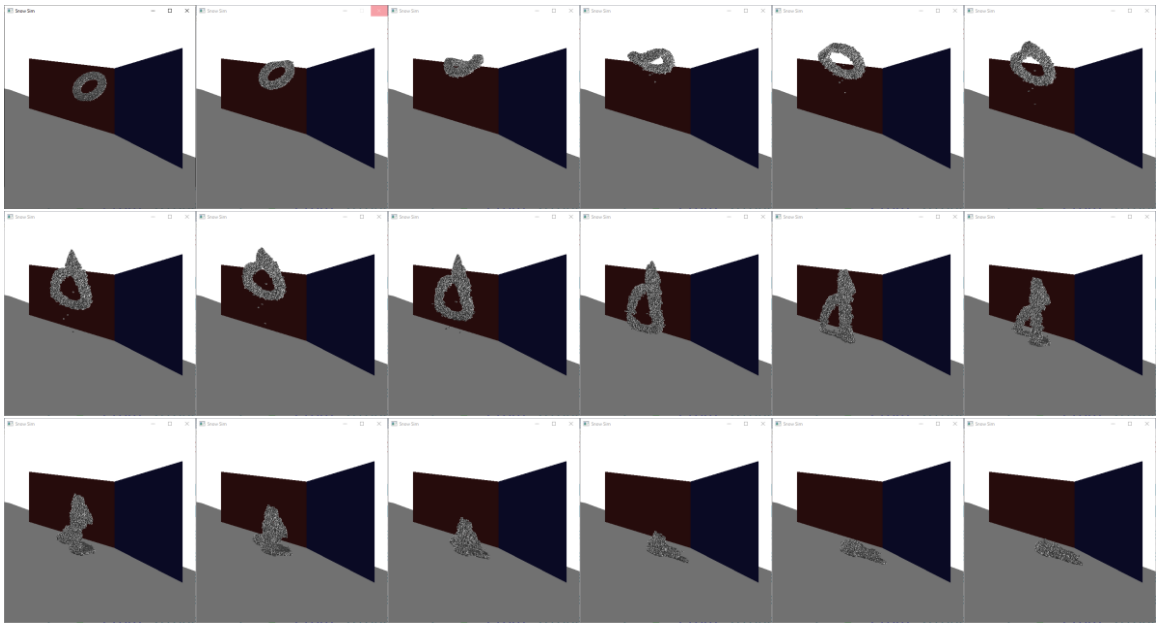
The overall program executes at approximately 19-20 FPS with 10k particles and a grid size of 100x100x100. A wavefront object loader was also created to load in mesh to be sampled for particles. To obtain the starting position and mass of each particle, the mesh is voxelized and uniformly sampled via rejection sampling methods. The volume is calculated and the particle's mass is set to $m_p = target\_density * \frac{volume}{num\_particles}$. Results of the simulation are shown in Diagrams 1 and 2 with snow particles represented as cubes.

**Diagram 1**



100k snow particles from a monkey head mesh are dropped on a moving ball. Screenshots are taking every five-frame interval.

**Diagram 2**



100k snow particles from a torus mesh hitting the top wall and sliding (the wall is not rendered). Screenshots are taken every five-frame interval.