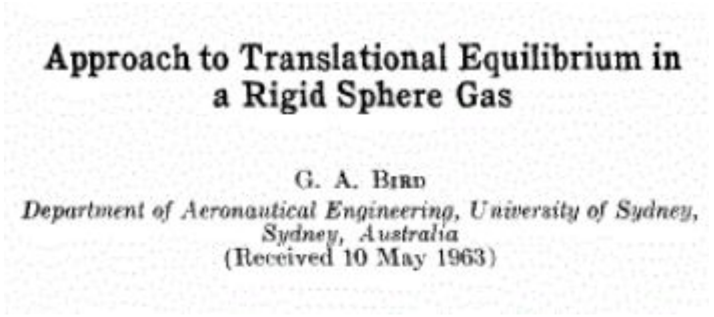
Simulating a dilute gas using DSMC

By -
Arnav Bhavar (19026)
Harsh Mishra (19049)
Rakshith M (19099)

How does one simulate a dilute gas?

Direct Simulation Monte Carlo (DSMC)!

Problem Statement


Approach to Translational Equilibrium in a Rigid Sphere Gas

G. A. Bird

Department of Aeronautical Engineering, University of Sydney, Sydney, Australia

(Received 10 May 1963)

- Dilute gas: Mean free path λ of the molecules is of the same order (or greater) than a representative physical length scale L in the problem.
- Cannot be accurately simulated using fluid (Navier-Stokes) equations.
- **Solution:** DSMC, a numerical method for modeling rarefied gas flows originally developed by Graeme. A. Bird in the 1960s.

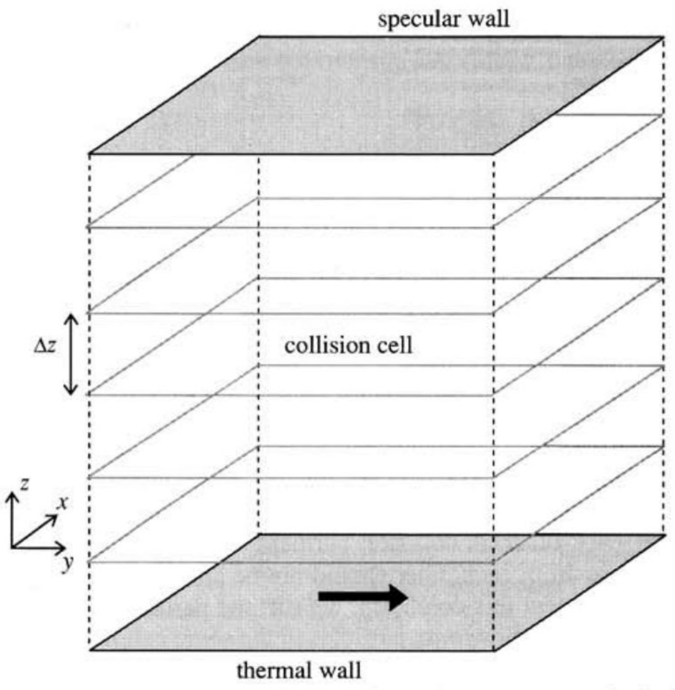
How does DSMC characterize flow?

1. A useful parameter for our discussion is the Knudsen number (Kn).
2. Kn=λ/L, so DSMC is used only in the regime Kn>1/10.
3. Rather than calculating all the collisions explicitly (molecular dynamics), DSMC generates collisions stochastically.
4. It gets the scattering rates and post-collision velocity distributions from the Kinetic theory of gases.
5. Mean free path from the Kinetic theory is:

$$\lambda = \frac{1}{\sqrt{2}\pi\sigma^2 n}$$

6. Hence, we will solve the Boltzmann equation for finite Knudsen number.

The Setup



Top plate: Reflective wall which will simply bounce back particles that hit it.

Bottom plate: Thermal wall with temperature $Tw=T_0$

Initial condition: Particles are distributed randomly in space. Particles are given velocities from a Maxwellian distribution.

What do the particles do in each time step?

Each particle does one or more of the following things:

1. It drifts.
2. It collides with a wall.
3. It collides with another particle.

If we have the initial conditions:

Position $r_i = [\ x_i, y_i, z_i\ ]$ and Velocity $v_i = [\ vx_i, vy_i, vz_i\ ]$

How do we take account of these things in our simulation?

1. Drift

Velocity cause drift hence we just update r_i (position of each particle) as:

$$\mathbf{r}_i = \mathbf{r}_i + \Delta t \times \mathbf{v}_i$$

Δt is the timestep in our simulation

2. Wall collisions

If a particle collides with the reflective wall, the normal component of the velocity is reversed.

If it collides the thermal wall, all components of the velocity are reset according to a Maxwellian distribution.

3. Particle-particle collisions

Each time step processes a set of representative collisions in each "collision" cell. Randomly chosen particle pairs i and j within the cell collide with a probability that is proportional to their relative velocity:

$$p_{\text{collide}}(i,j) \propto |\mathbf{v}_i - \mathbf{v}_j|$$

Whether to accept or reject the pair for collision?

For each cell, we randomly select M_{cand} candidate pairs for collision

$$M_{\text{cand}} = \frac{N_c^2 \pi \sigma^2 v_{\text{r,max}} N_e \Delta t}{2V_c}$$

where

N_c is the number of particles in the cell

V_c is the volume of the cell, and

$v_{\text{r,max}}$ is an upper limit estimate for the maximum relative velocity between particles.

We **accept** the pair for collision if for any random number r between 0 and 1, the following is satisfied:

$$\frac{|\mathbf{v}_i - \mathbf{v}_j|}{v_{\mathrm{r,max}}} > r$$

The expected number of collisions M_{coll} out of the M_{cand} candidate pairs is:

$$\frac{M_{\mathrm{coll}}}{M_{\mathrm{cand}}} = \frac{\langle v_{\mathrm{r}} \rangle}{v_{\mathrm{r,max}}}$$

Where $<v_r>$ is the average relative velocity of the particles in the cell

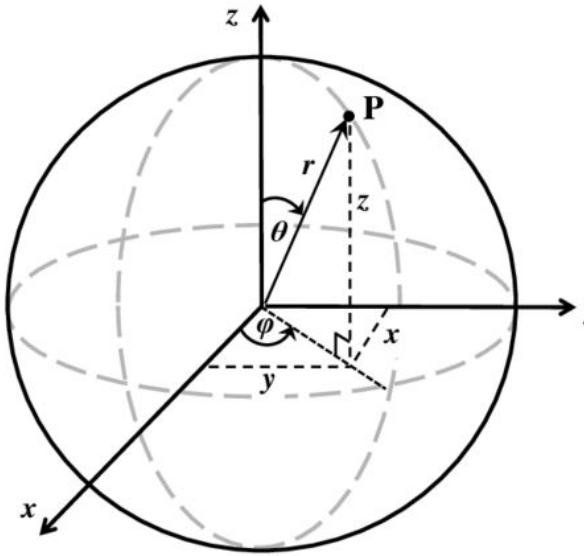
After collision, the particle velocities of particles i and j are changed to

$$\mathbf{v}'_i = \mathbf{v}_{\mathrm{cm}} + \frac{1}{2}\mathbf{v}'_{\mathrm{r}} \qquad \mathbf{v}'_j = \mathbf{v}_{\mathrm{cm}} - \frac{1}{2}\mathbf{v}'_{\mathrm{r}}$$

where

$$\mathbf{v}_{\mathrm{cm}} = \frac{1}{2}(\mathbf{v}_i + \mathbf{v}_j)$$

$$\mathbf{v}'_{\mathrm{r}} = v_r \left[\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta\right]$$



$x = r \sin\theta \cos\varphi$

$y = r \sin\theta \sin\varphi$

$z = r \cos\theta$

$r = \sqrt{x^2 + y^2 + z^2}$

$\theta = \cos^{-1}(z/r)$

$\varphi = \tan^{-1}(y/x)$

- We perform the sampling such that it uniformly selects a random direction on a sphere's surface.
- V_r’ is the randomly re-oriented relative velocity between the two particles:
- Let r and r’ be random numbers drawn from the interval (0,1).
- The angles and then chosen randomly as:

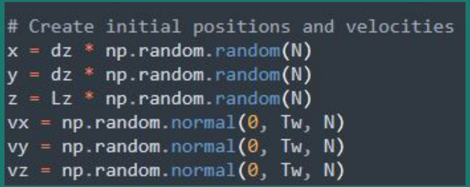
$$\cos\theta = 2r - 1$$
$$\sin\theta = \sqrt{1 - \cos^2\theta}$$
$$\phi = 2\pi r'$$

CODING ALGORITHM

1

Initial position and velocities

We first define initial positions and velocities of the particles

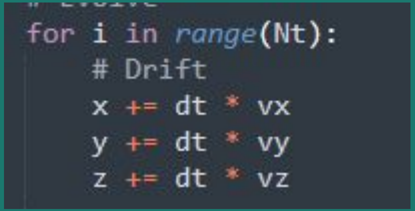


```
# Create initial positions and velocities
x = dz * np.random.random(N)
y = dz * np.random.random(N)
z = Lz * np.random.random(N)
vx = np.random.normal(0, Tw, N)
vy = np.random.normal(0, Tw, N)
vz = np.random.normal(0, Tw, N)
```

2

Drift velocity

We add drift velocity to the particles and calculate new position



```
for i in range(Nt):
    # Drift
    x += dt * vx
    y += dt * vy
    z += dt * vz
```

3

Collision between wall and particles

We add collision between the particles by a method that if z coordinate of particle is greater than height of box, it's velocity reverses

4

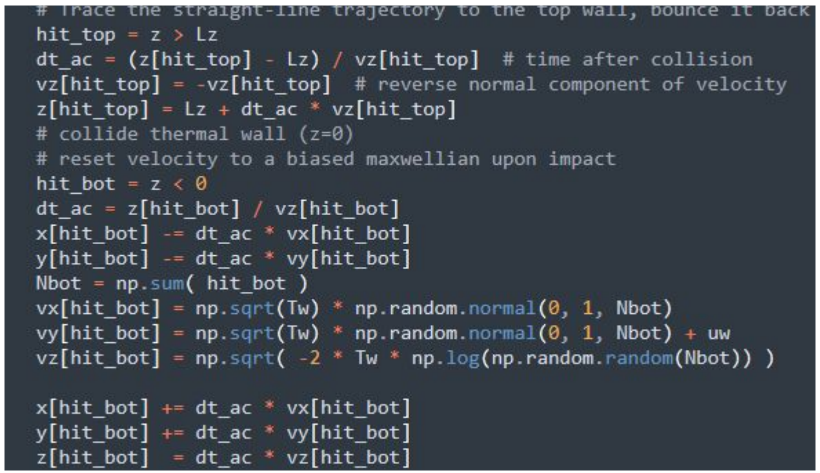
Collision between Particles

We collide particles using acceptance--rejection scheme

Code snippets in coming slides for 3 & 4

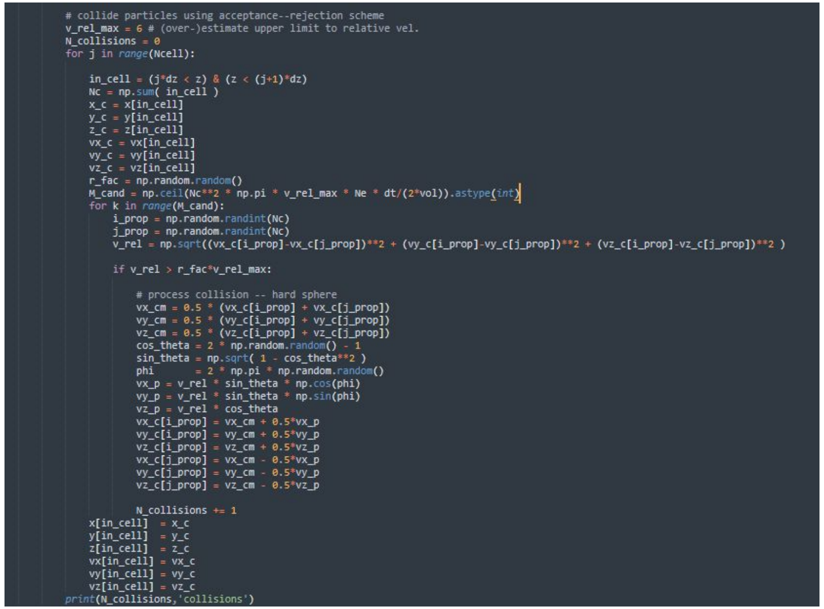
Code Snippets for collision

Collision with walls



```python
# Trace the straight-line trajectory to the top wall, bounce it back
hit_top = z > Lz
dt_ac = (z[hit_top] - Lz) / vz[hit_top]  # time after collision
vz[hit_top] = -vz[hit_top]  # reverse normal component of velocity
z[hit_top] = Lz + dt_ac * vz[hit_top]
# collide thermal wall (z=0)
# reset velocity to a biased maxwellian upon impact
hit_bot = z < 0
dt_ac = z[hit_bot] / vz[hit_bot]
x[hit_bot] -= dt_ac * vx[hit_bot]
y[hit_bot] -= dt_ac * vy[hit_bot]
Nbot = np.sum( hit_bot )
vx[hit_bot] = np.sqrt(Tw) * np.random.normal(0, 1, Nbot)
vy[hit_bot] = np.sqrt(Tw) * np.random.normal(0, 1, Nbot) + uw
vz[hit_bot] = np.sqrt( -2 * Tw * np.log(np.random.random(Nbot)) )

x[hit_bot] += dt_ac * vx[hit_bot]
y[hit_bot] += dt_ac * vy[hit_bot]
z[hit_bot]  = dt_ac * vz[hit_bot]
```

Collision of particles with each other

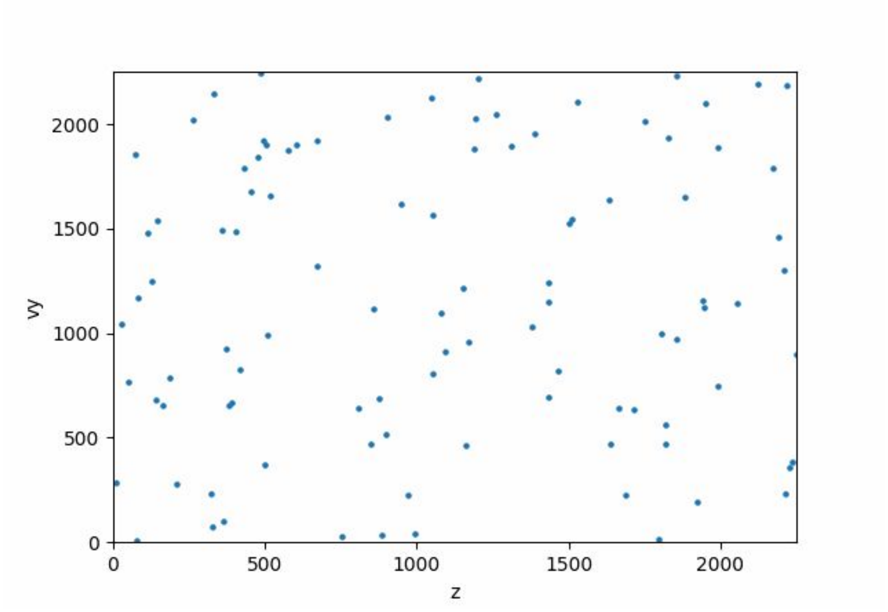


```python
# collide particles using acceptance--rejection scheme
v_rel_max = 6 # (over-)estimate upper limit to relative vel.
N_collisions = 0
for j in range(Ncell):

    in_cell = (j*dz < z) & (z < (j+1)*dz)
    Nc = np.sum( in_cell )
    x_c = x[in_cell]
    y_c = y[in_cell]
    z_c = z[in_cell]
    vx_c = vx[in_cell]
    vy_c = vy[in_cell]
    vz_c = vz[in_cell]
    r_fac = np.random.random()
    M_cand = np.ceil(Nc**2 * np.pi * v_rel_max * Ne * dt/(2*vol)).astype(int)
    for k in range(M_cand):
        i_prop = np.random.randint(Nc)
        j_prop = np.random.randint(Nc)
        v_rel = np.sqrt((vx_c[i_prop]-vx_c[j_prop])**2 + (vy_c[i_prop]-vy_c[j_prop])**2 + (vz_c[i_prop]-vz_c[j_prop])**2 )

        if v_rel > r_fac*v_rel_max:

            # process collision -- hard sphere
            vx_cm = 0.5 * (vx_c[i_prop] + vx_c[j_prop])
            vy_cm = 0.5 * (vy_c[i_prop] + vy_c[j_prop])
            vz_cm = 0.5 * (vz_c[i_prop] + vz_c[j_prop])
            cos_theta = 2 * np.random.random() - 1
            sin_theta = np.sqrt( 1 - cos_theta**2 )
            phi       = 2 * np.pi * np.random.random()
            vx_p = v_rel * sin_theta * np.cos(phi)
            vy_p = v_rel * sin_theta * np.sin(phi)
            vz_p = v_rel * cos_theta
            vx_c[i_prop] = vx_cm + 0.5*vx_p
            vy_c[i_prop] = vy_cm + 0.5*vy_p
            vz_c[i_prop] = vz_cm + 0.5*vz_p
            vx_c[j_prop] = vx_cm - 0.5*vx_p
            vy_c[j_prop] = vy_cm - 0.5*vy_p
            vz_c[j_prop] = vz_cm - 0.5*vz_p

            N_collisions += 1
    x[in_cell]  = x_c
    y[in_cell]  = y_c
    z[in_cell]  = z_c
    vx[in_cell] = vx_c
    vy[in_cell] = vy_c
    vz[in_cell] = vz_c
print(N_collisions,'collisions')
```

RESULTS

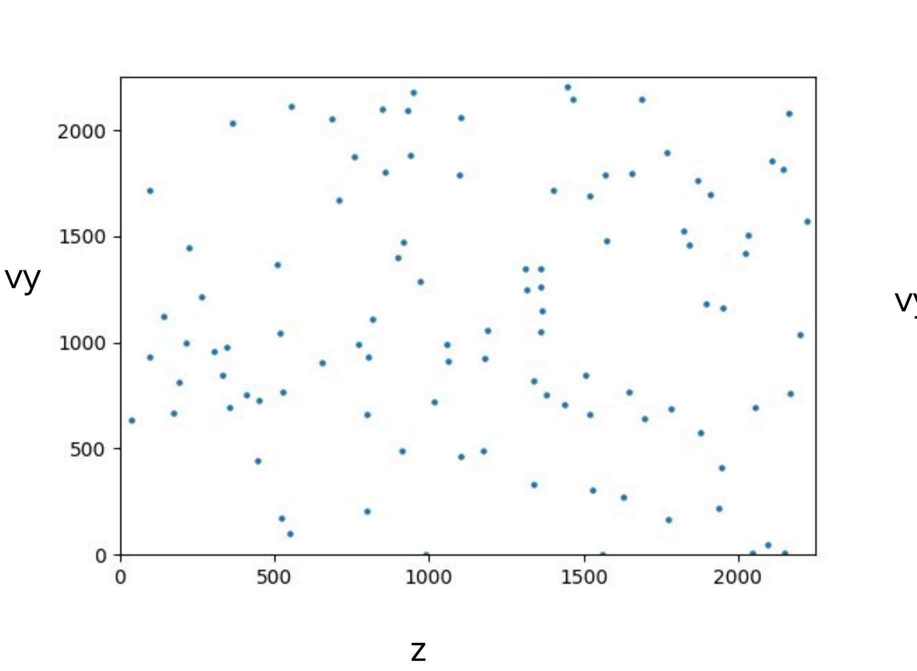
This is essentially a phase-space slice of the gas in real time.

100 PARTICLES

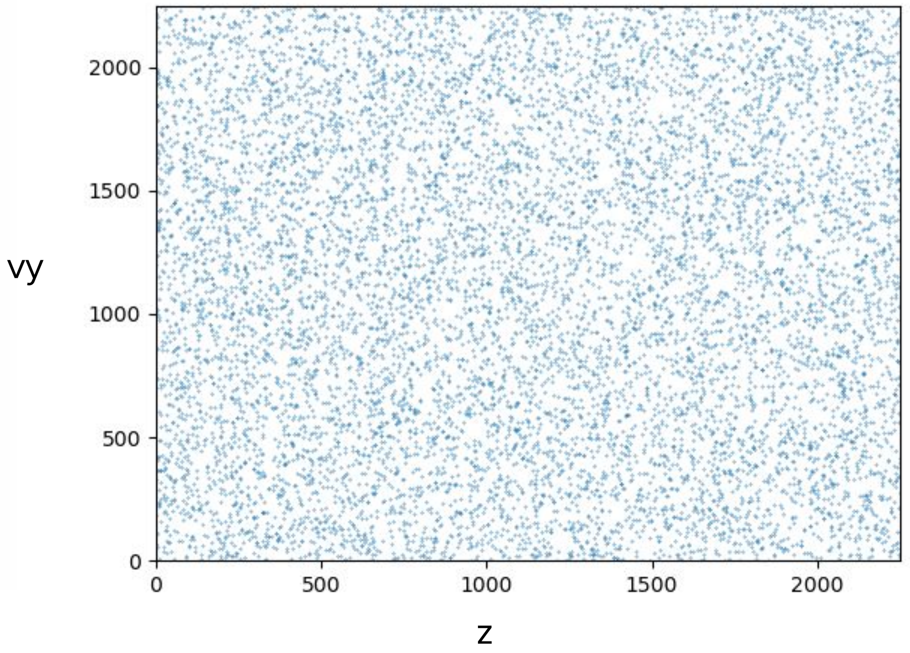


Increasing the number of particles

N=100

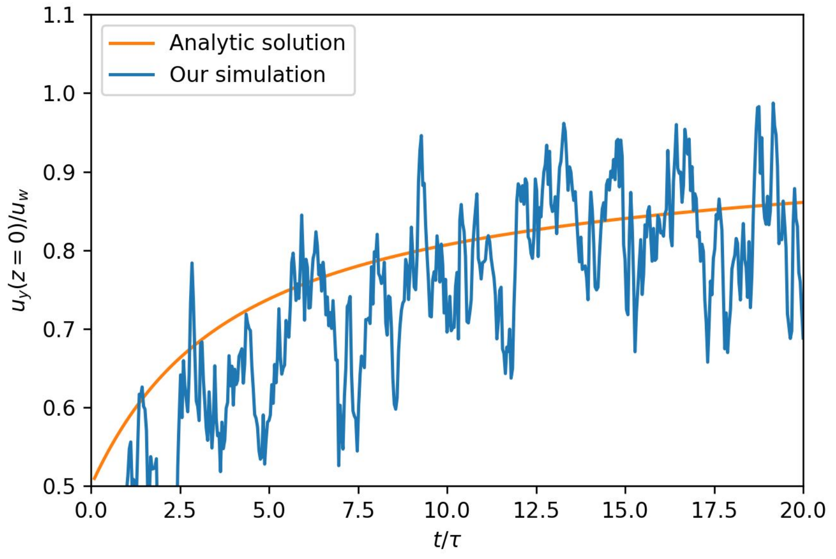


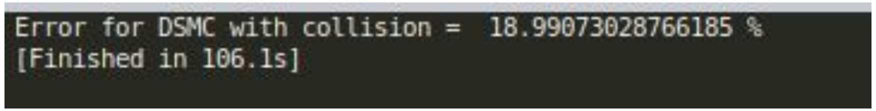
N=10,000



Comparison with theory

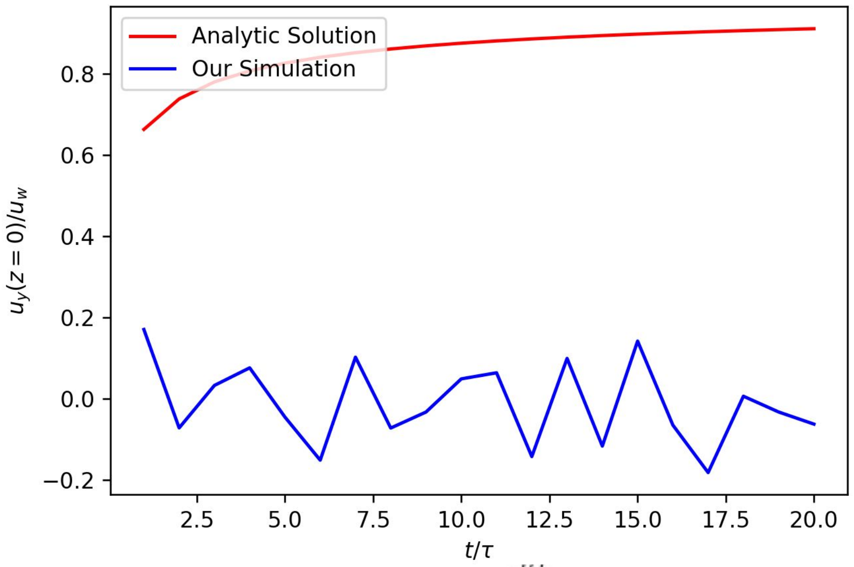
Comparison for 50000 particles with collision

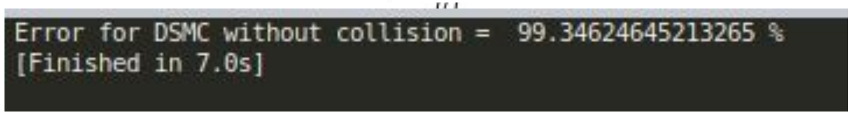




```
Error for DSMC with collision =  18.99073028766185 %
[Finished in 106.1s]
```

Comparison for 50000 particles without collision





```
Error for DSMC without collision =  99.34624645213265 %
[Finished in 7.0s]
```

Applications

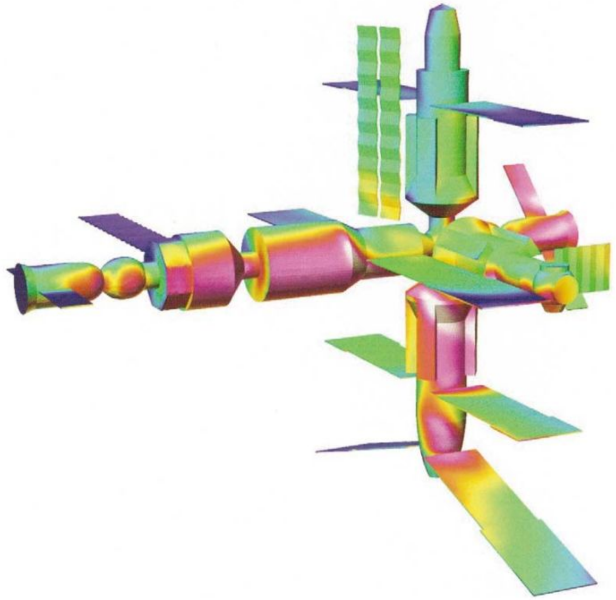


Fig. 6. DAC predicted surface pressure on Mir from Shuttle Norm-Z jets at 5 m from docking.

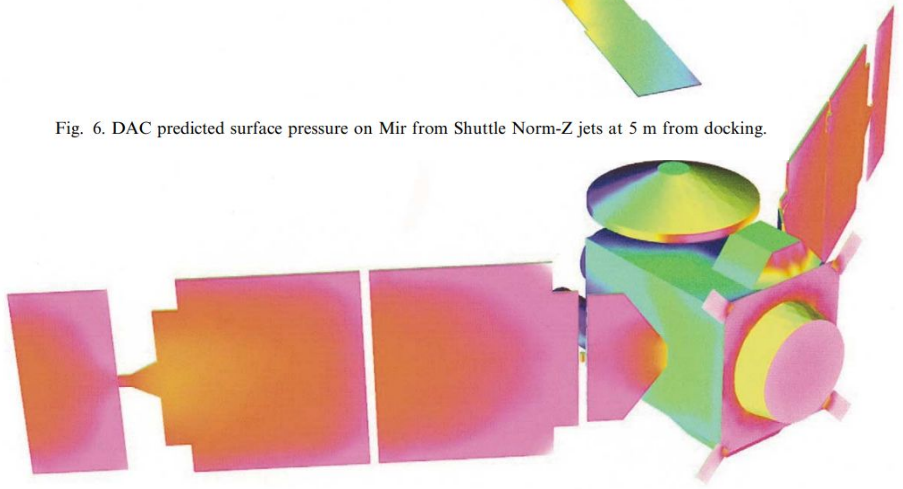
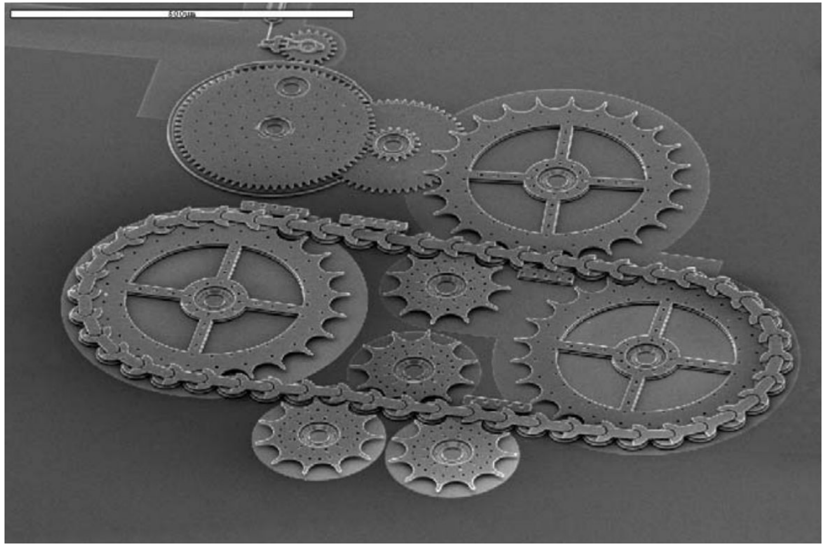


Fig. 10. Surface heating distribution for Mars Global Surveyor vehicle calculated using DAC.

NASA uses the DAC (DSMC Analysis Code) to determine aerodynamic and heating behavior of satellites and shuttles in different situations like re-entry and docking. This is especially useful when experimental reproduction of flow regime is impossible.

Reference: LeBeau, G. J., & Lumpkin, F. E. (2001)

Gas effects on moving microdevices



Gas surrounding microdevices can affect their performance.

Since these devices are in micron length scales, gas flow cannot be modelled using Navier Stokes.

They are also typically packed under low pressure conditions.

DSMC is used to check if gas causes:

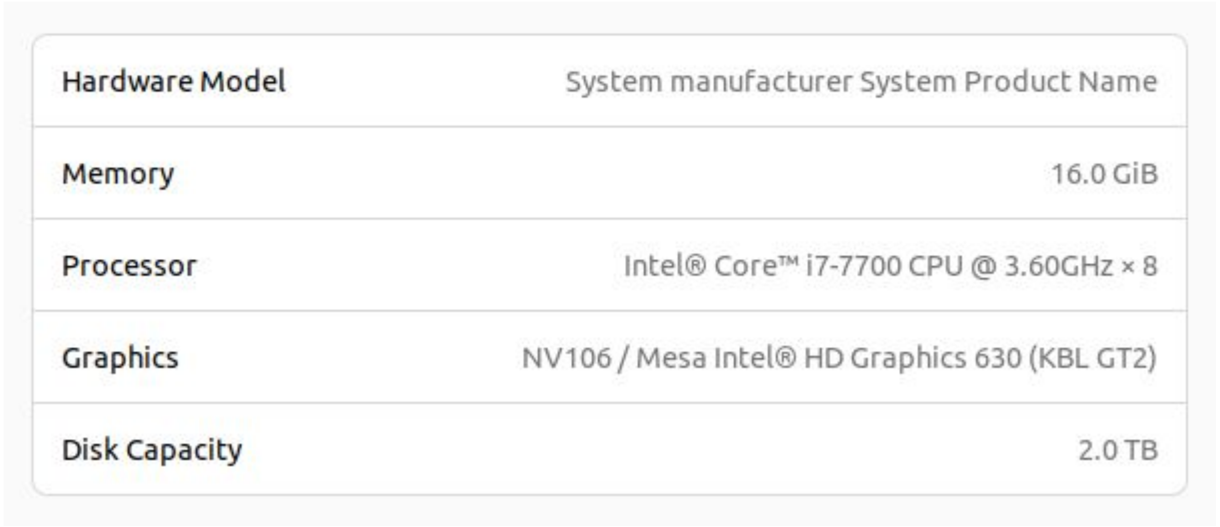
- Increase in power consumption.
- Frequency change.
- Long-term damage.

Simulation of the Volcano on Jupiter's moon, Io.

Limitations and sources of error

- The computational load increases significantly with the density of the flow.
- DSMC can carry more information than actually needed for some applications, hence can be inefficient.
- The accuracy of the simulation results depends on the number of particles used in the simulation. In low-density regions, statistical noise can be significant, leading to inaccurate results.
- The size of the time step affects the accuracy of the simulation. If the time step is too large, fast-changing phenomena may be inaccurately captured, leading to errors.
- DSMC typically divides the computational domain into a grid of cells to track the particles' positions and velocities. The choice of grid size can impact the accuracy of the simulation.

PC details



Hardware Model	System manufacturer System Product Name
Memory	16.0 GiB
Processor	Intel® Core™ i7-7700 CPU @ 3.60GHz × 8
Graphics	NV106 / Mesa Intel® HD Graphics 630 (KBL GT2)
Disk Capacity	2.0 TB

Time taken for simulation

100 particles, 40fps: 16.6 s ± 629 ms

500 particles, 17.7 s ± 542 ms

10000 particles 19.1 ± 172 ms

References:

1. G. A. Bird; Approach to Translational Equilibrium in a Rigid Sphere Gas. Physics of Fluids 1 October 1963; 6 (10): 1518–1519. https://doi.org/10.1063/1.1710976
2. Alexander, Francis J. and Garcia, Alejandro L. "The Direct Simulation Monte Carlo Method." Computers in Physics v11 n 6, 1997.
3. https://en.wikipedia.org/wiki/Kinetic_theory_of_gases
4. https://en.wikipedia.org/wiki/Direct_simulation_Monte_Carlo
5. LeBeau, G. J., & Lumpkin, F. E. (2001). Application highlights of the DSMC Analysis Code (DAC) software for simulating rarefied flows. Computer Methods in Applied Mechanics and Engineering, 191(6–7), 595–609. https://doi.org/10.1016/s0045-7825(01)00304-8