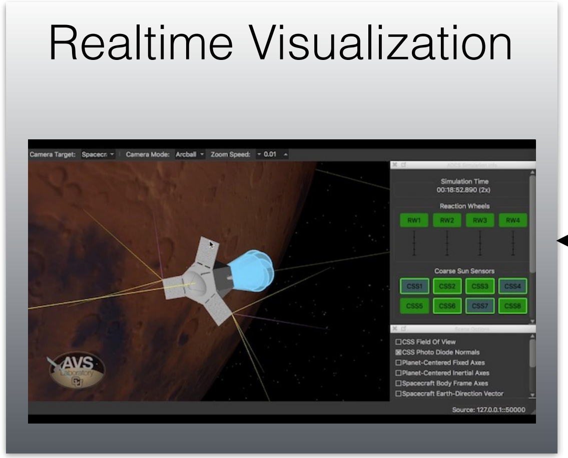
Incorporating Planetary Space Environments into the Basilisk Astrodynamics Framework

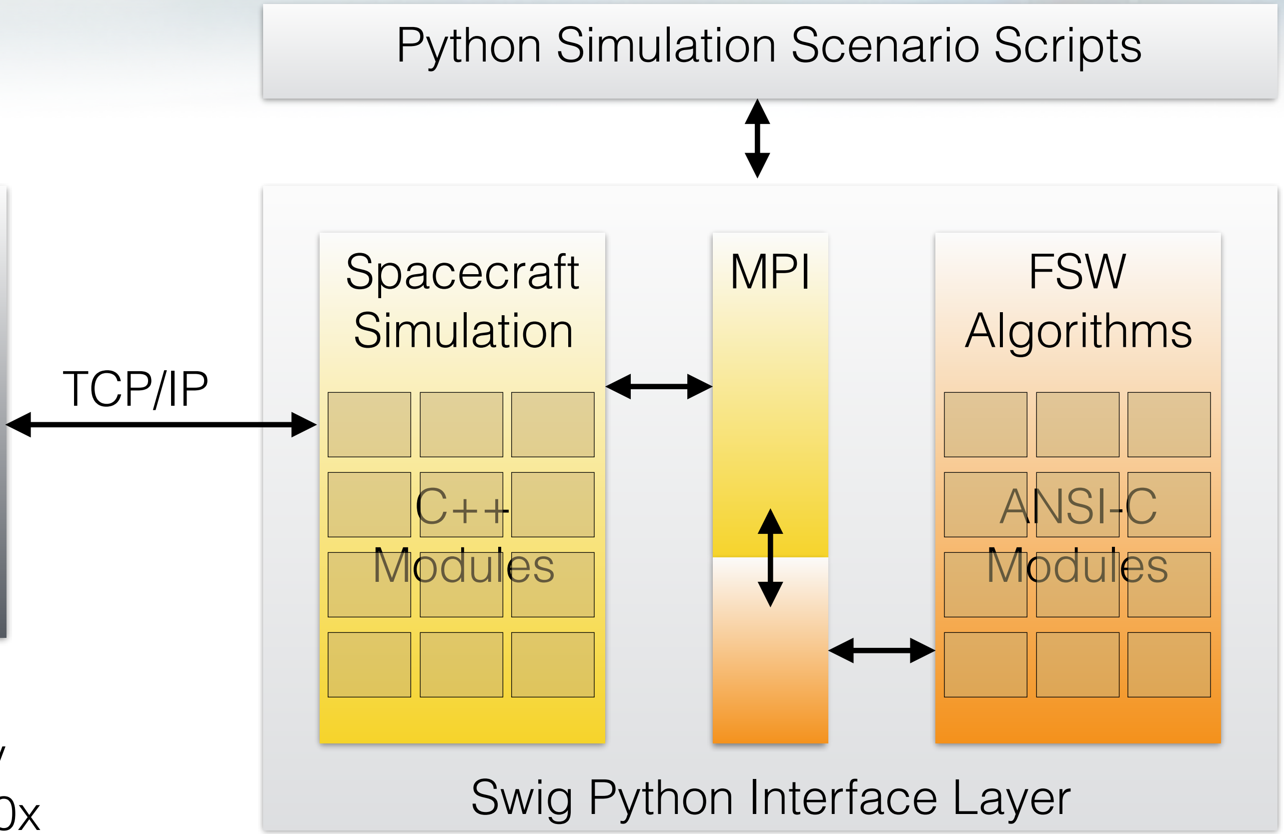
Hanspeter Schaub, *Professor, Glenn L. Murphy Chair of Engineering*
Patrick Kenneally, *Software Systems Architect, Mission Modeling and Verification, JPL*
Andrew Harris, *Graduate Research Assistant*

Advances and Applications of Computational Astrodynamics , APCOM 2019, Taipei, Taiwan

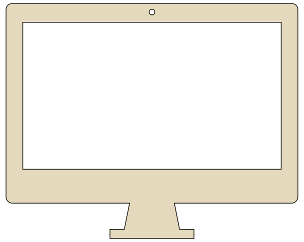
Ann and H. J. Smead Aerospace
Engineering Sciences Department
University of Colorado, **Boulder**

Basilisk Software Architecture



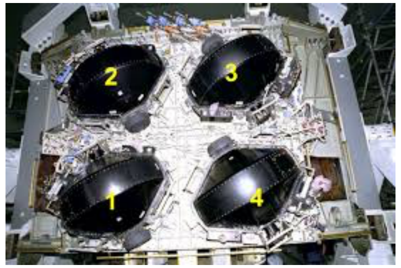


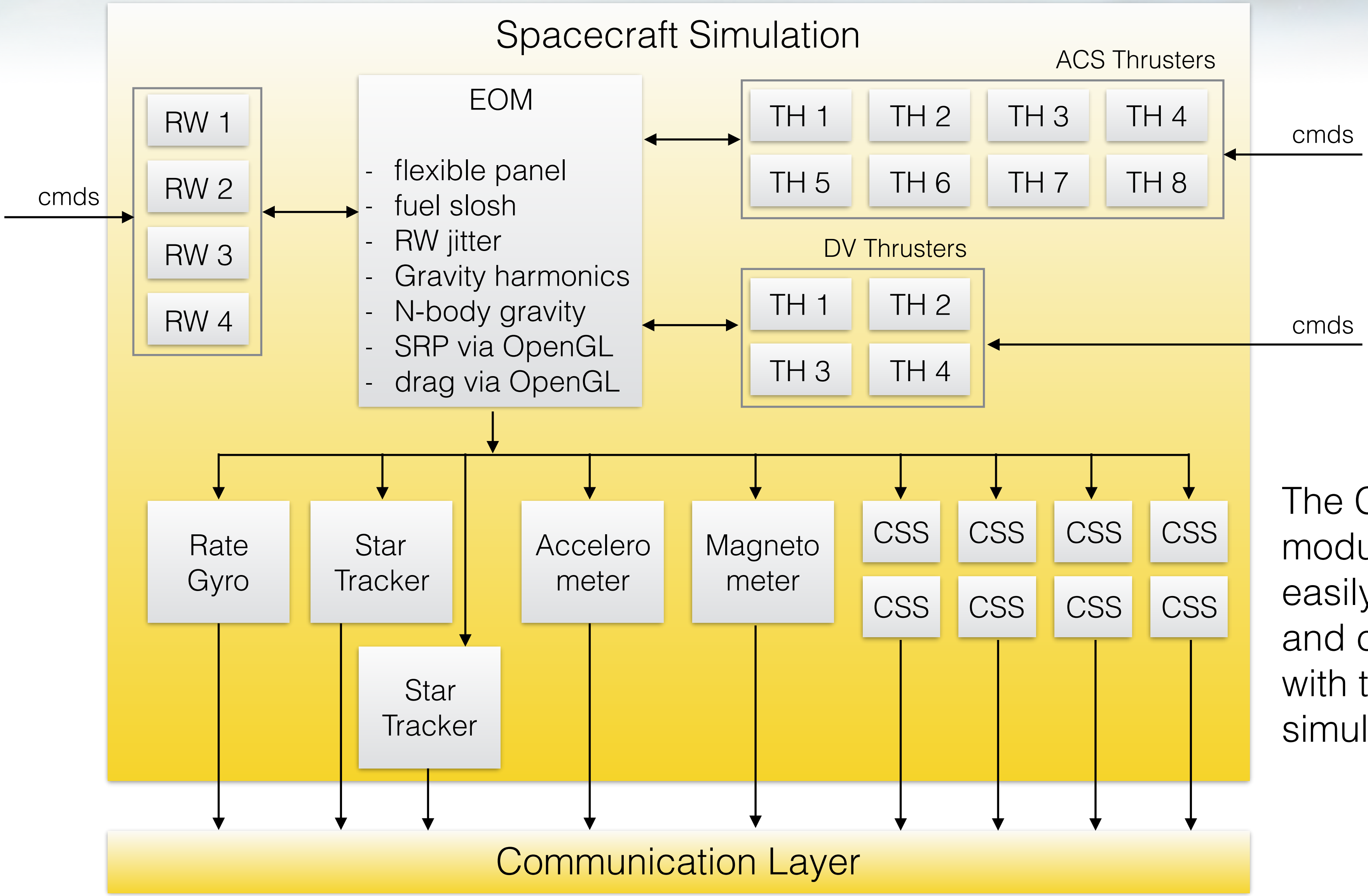
- Monte-Carlo Capability
- Speeds up to 700-1000x
- Software Realtime Mode
- ISC Open Source License



P. Kenneally, H. Schaub and S. Piggott, “Basilisk: A Flexible, Scalable and Modular Astrodynamics Simulation Framework,” 7th International Conference on Astrodynamics Tools and Techniques (ICATT), DLR Oberpfaffenhofen, Germany, November 6–9, 2018.

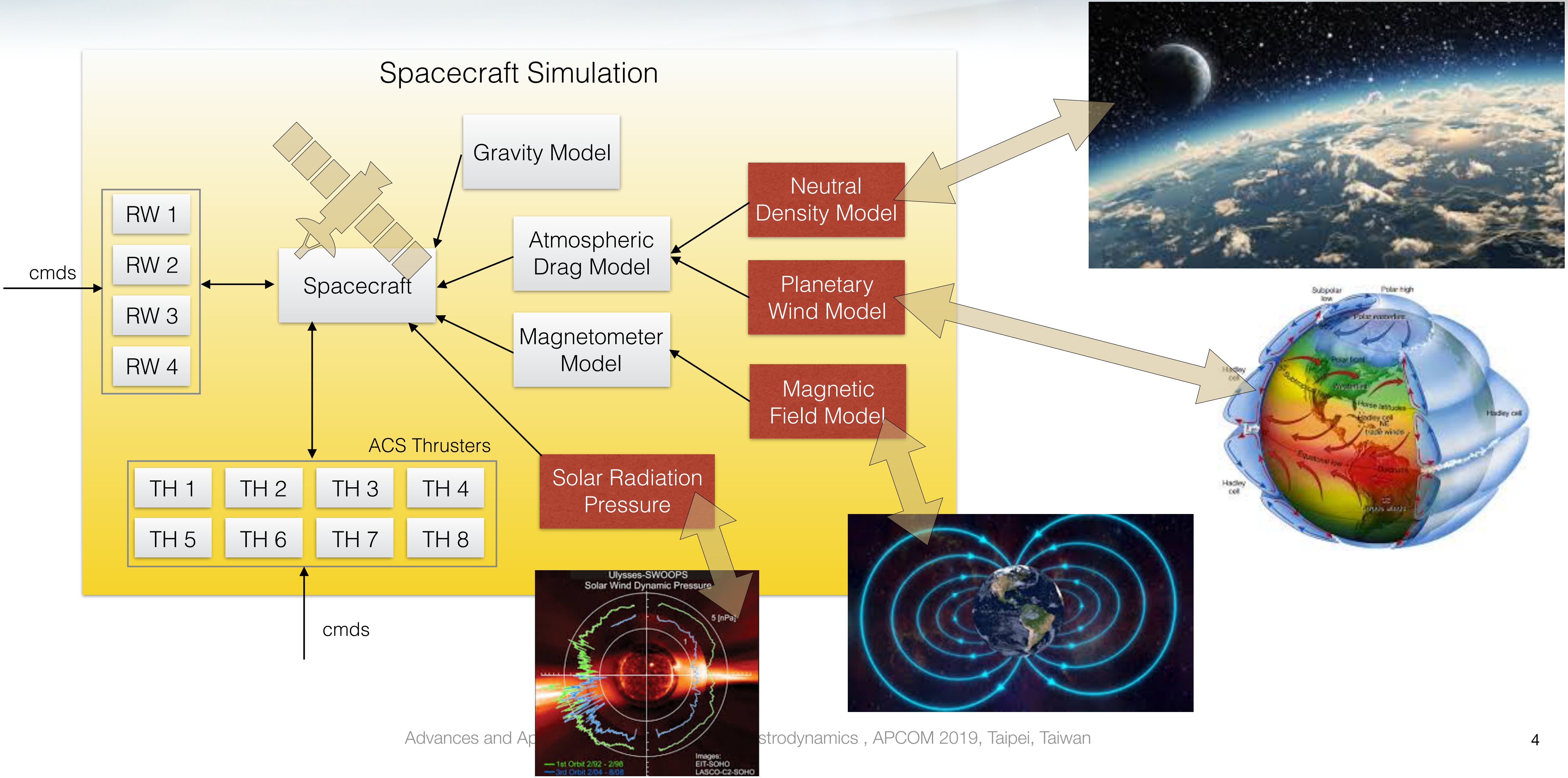
Sample Spacecraft Simulation Setup





The C++ device modules can easily be added and connected with the the simulation.

Spacecraft Environment Integration



Earth Environment Modeling

Magnetic field Models:

- centered dipole model
- WMM

Neutral Density Models:

- exponential atmosphere
- MSIS

MagneticFieldBaseClass: I/O Messages

MagneticFieldBaseClass: Methods

```
class MagneticFieldBase: public SysModel  {

public:
    void SelfInit();
    void CrossInit();
    void Reset(uint64_t CurrentSimNanos);
    void addSpacecraftToModel(std::string
                              tmpScMsgName);
    void UpdateState(uint64_t CurrentSimNanos);

    std::vector<std::string> scStateInMsgNames;
    std::vector<std::string> envOutMsgNames;
    std::string planetPosInMsgName;
    double envMinReach;
    double envMaxReach;
    double planetRadius;
```

```
protected:
    void writeMessages(uint64_t CurrentClock);
    bool readMessages();
    void updateLocalMagField(double currentTime);
    void updateRelativePos(SpicePlanetStateSimMsg
        *planetState, SCPlusStatesSimMsg *scState);

    virtual void evaluateMagneticFieldModel(
                    MagneticFieldSimMsg *msg,
                    double currentTime) = 0;
    virtual void customSelfInit();
    virtual void customCrossInit();
    virtual void customReset(uint64_t
                                  CurrentClock);
    virtual void customWriteMessages(uint64_t
                                  CurrentClock);
    virtual bool customReadMessages();
```

MagneticFieldCenteredDipole.cpp

```
void MagneticFieldCenteredDipole::evaluateMagneticFieldModel(MagneticFieldSimMsg *msg, double currentTime)
{
    Eigen::Vector3d magField_P;         // [T] magnetic field in Planet fixed frame
    Eigen::Vector3d rHat_P;             // [] normalized position vector in E frame components
    Eigen::Vector3d dipoleCoefficients; // [] The first 3 IGRF coefficient that define the magnetic dipole

    //! - compute normalized E-frame position vector
    rHat_P = this->r_BP_P.normalized();

    //! - compute magnetic field vector in E-frame components (see p. 405 in doi:10.1007/978-1-4939-0802-8)
    dipoleCoefficients << this->g11, this->h11, this->g10;
    magField_P = pow(this->planetRadius/this->orbitRadius,3)*(3* rHat_P*rHat_P.dot(dipoleCoefficients)
                 - dipoleCoefficients);

    //! - convert magnetic field vector in N-frame components and store in output message
    m33tMultV3(this->planetState.J20002Pfix, magField_P.data(), msg->magField_N);

    return;
}
```

MagneticFieldWMM

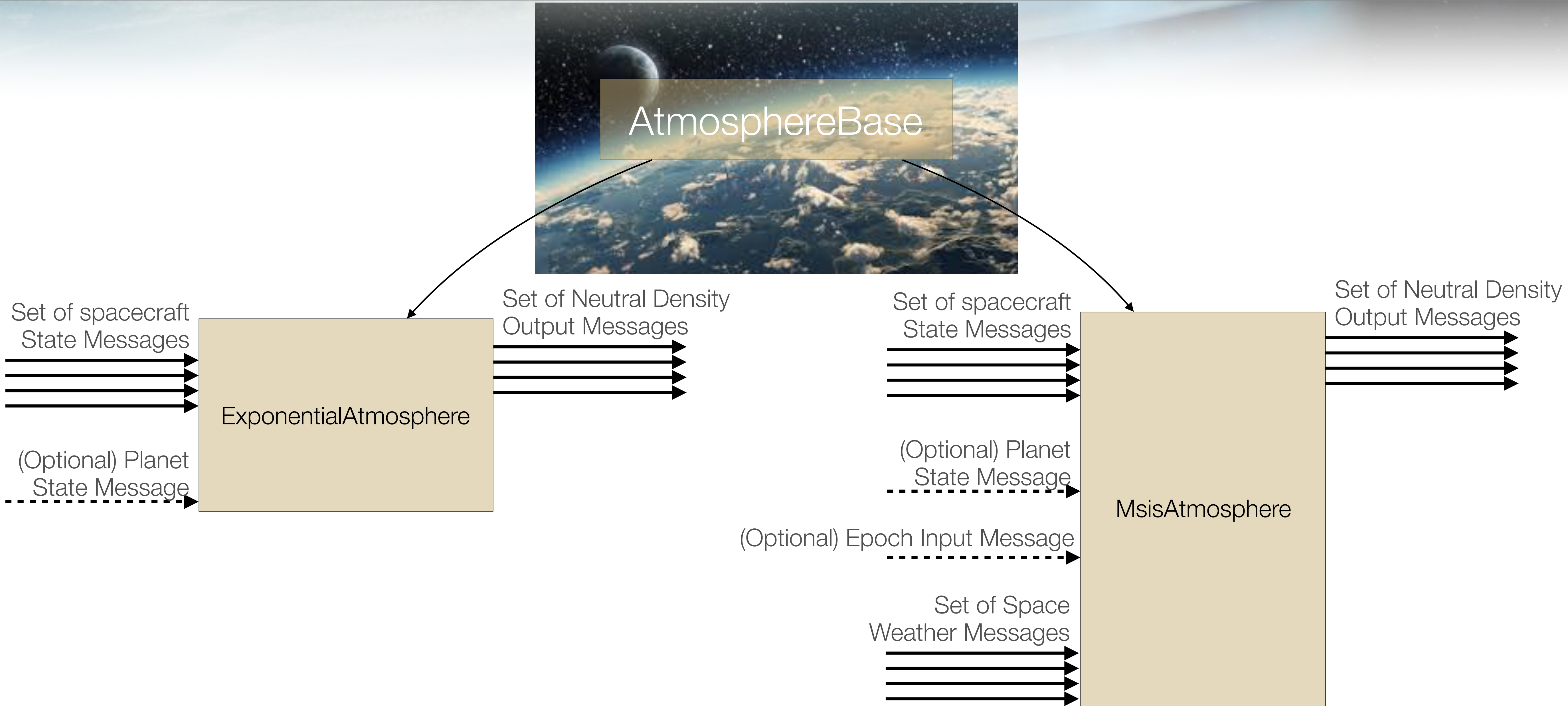
Epoch States:

- BSK default 2019-01-01, 00:00:00
- Set from Python directly within the module
- Read in through an optional epoch input message

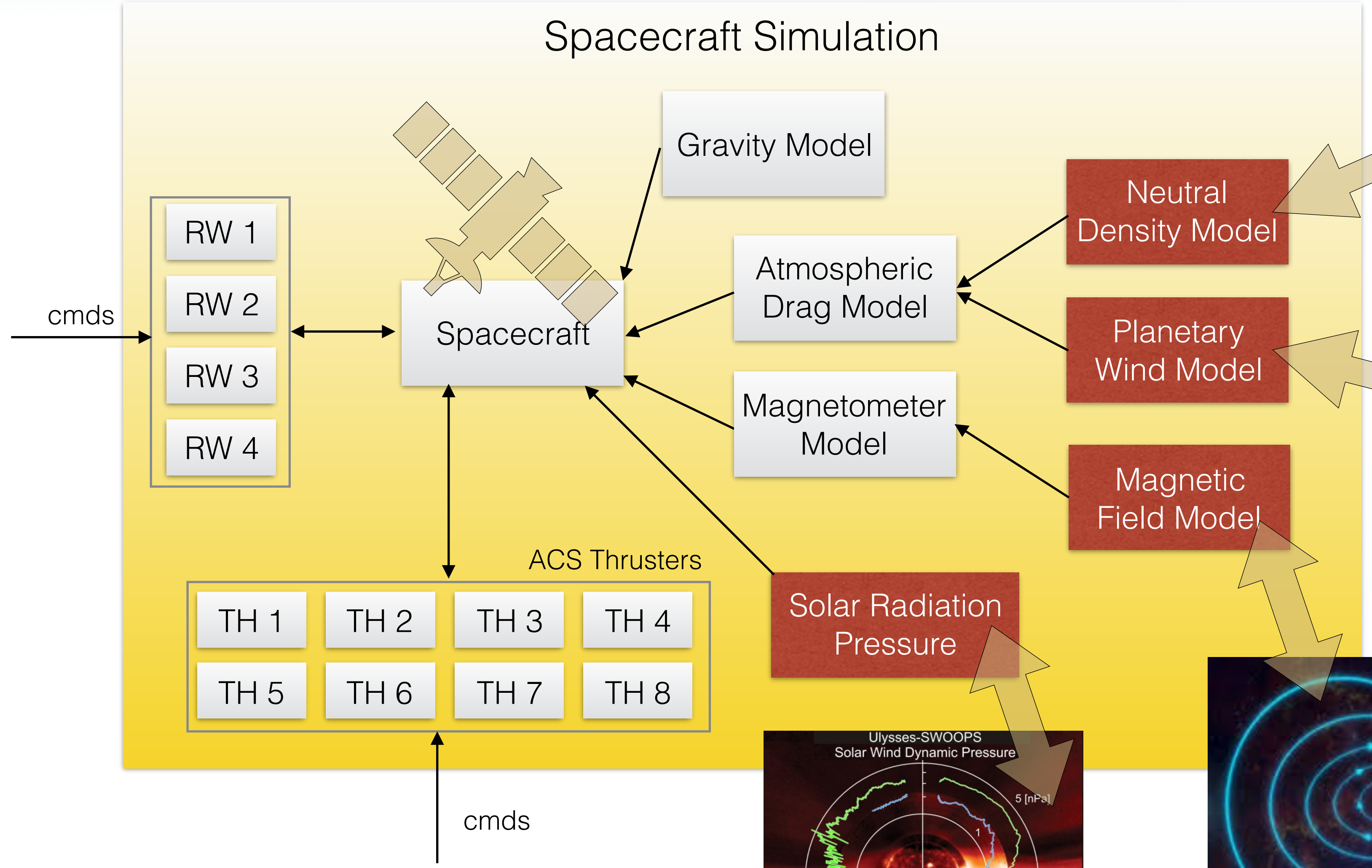
MagneticFieldWMM.h

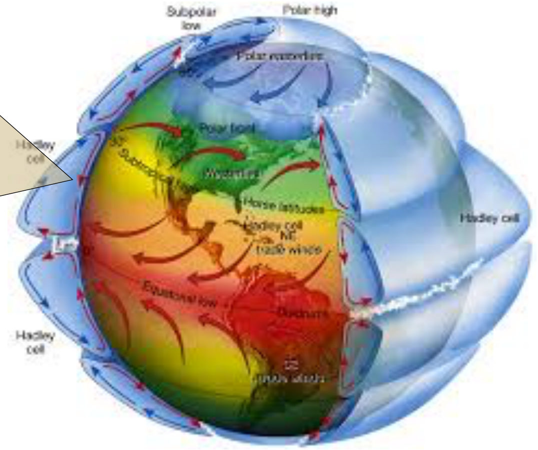
```
class MagneticFieldWMM:  public MagneticFieldBase {
public:
    MagneticFieldWMM();
    ~MagneticFieldWMM();

private:
    void evaluateMagneticFieldModel(MagneticFieldSimMsg *msg, double currentTime);

    void initializeWmm(const char *dataPath);
    void cleanupEarthMagFieldModel();
    void computeWmmField(double decimalYear, double phi, double lambda, double h, double B_M[3]);

    void customReset(uint64_t CurrentClock);
    void customCrossInit();
    void customSetEpochFromVariable();

    void decimalYear2Gregorian(double fractionalYear, struct tm *gregorian);
    double gregorian2DecimalYear(double currentTime);

public:
    std::string epochInMsgName;             //!< -- Message name of the epoch message
    std::string dataPath;                   //!< -- String with the path to the WMM coefficient file
    double      epochDateFractionalYear;    //!< Specified epoch date as a fractional year
```

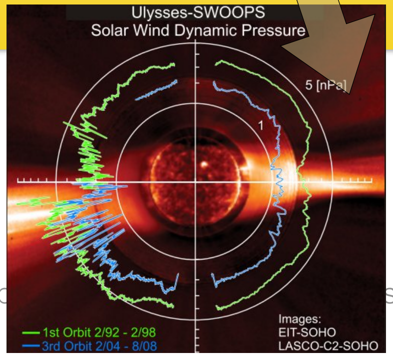
Neutral Atmospheric Density Modeling



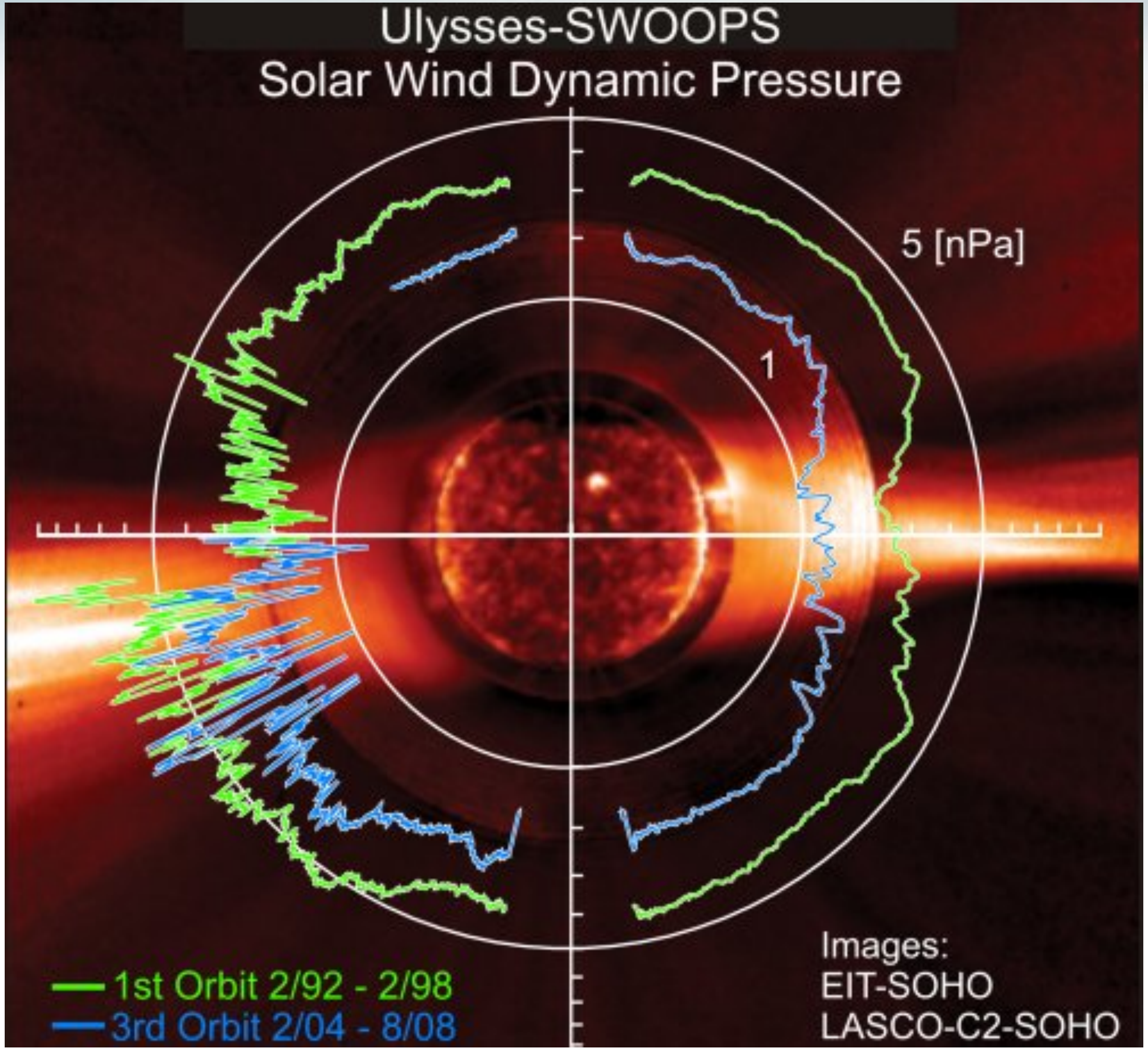
Spacecraft Environment Integration

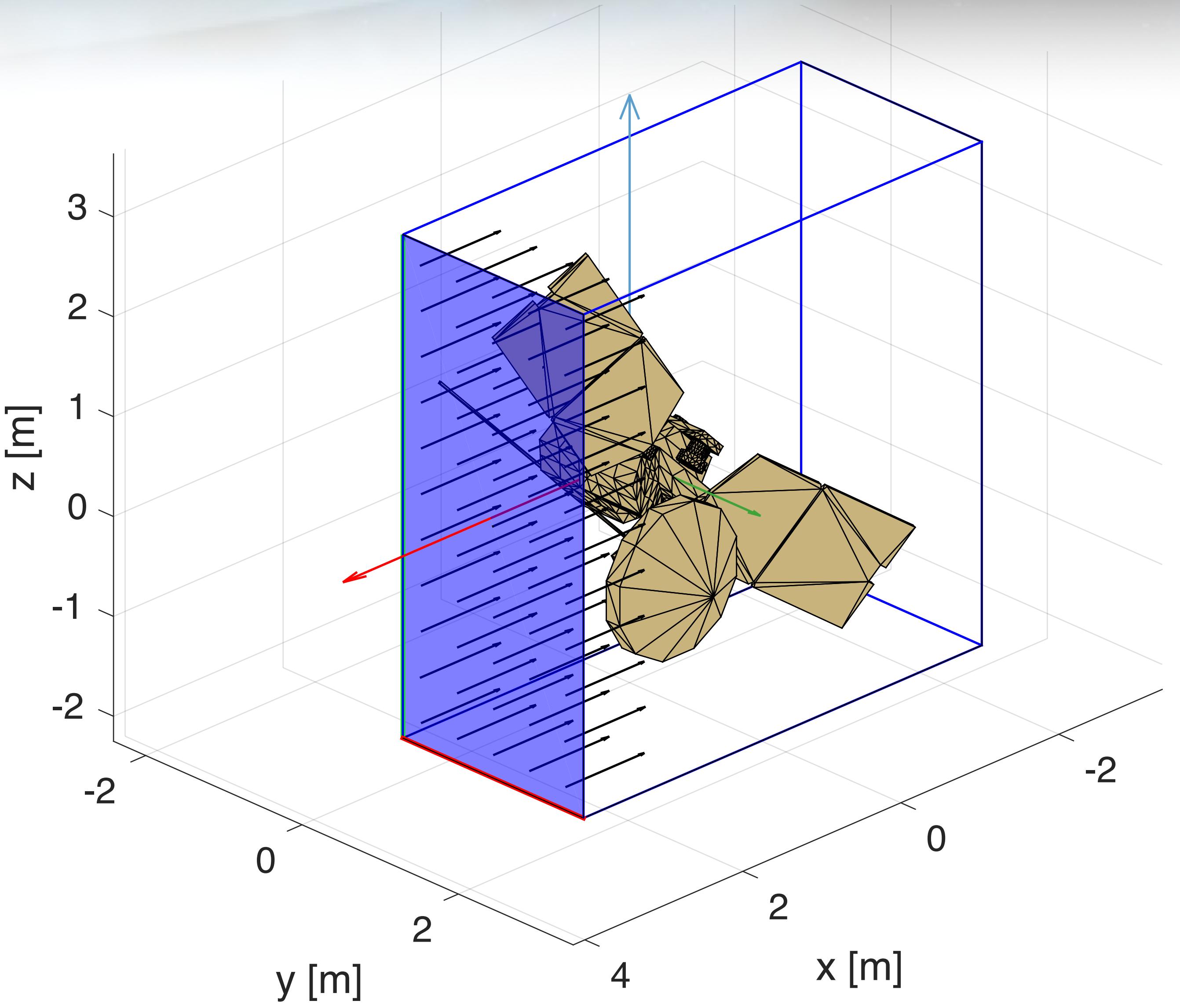






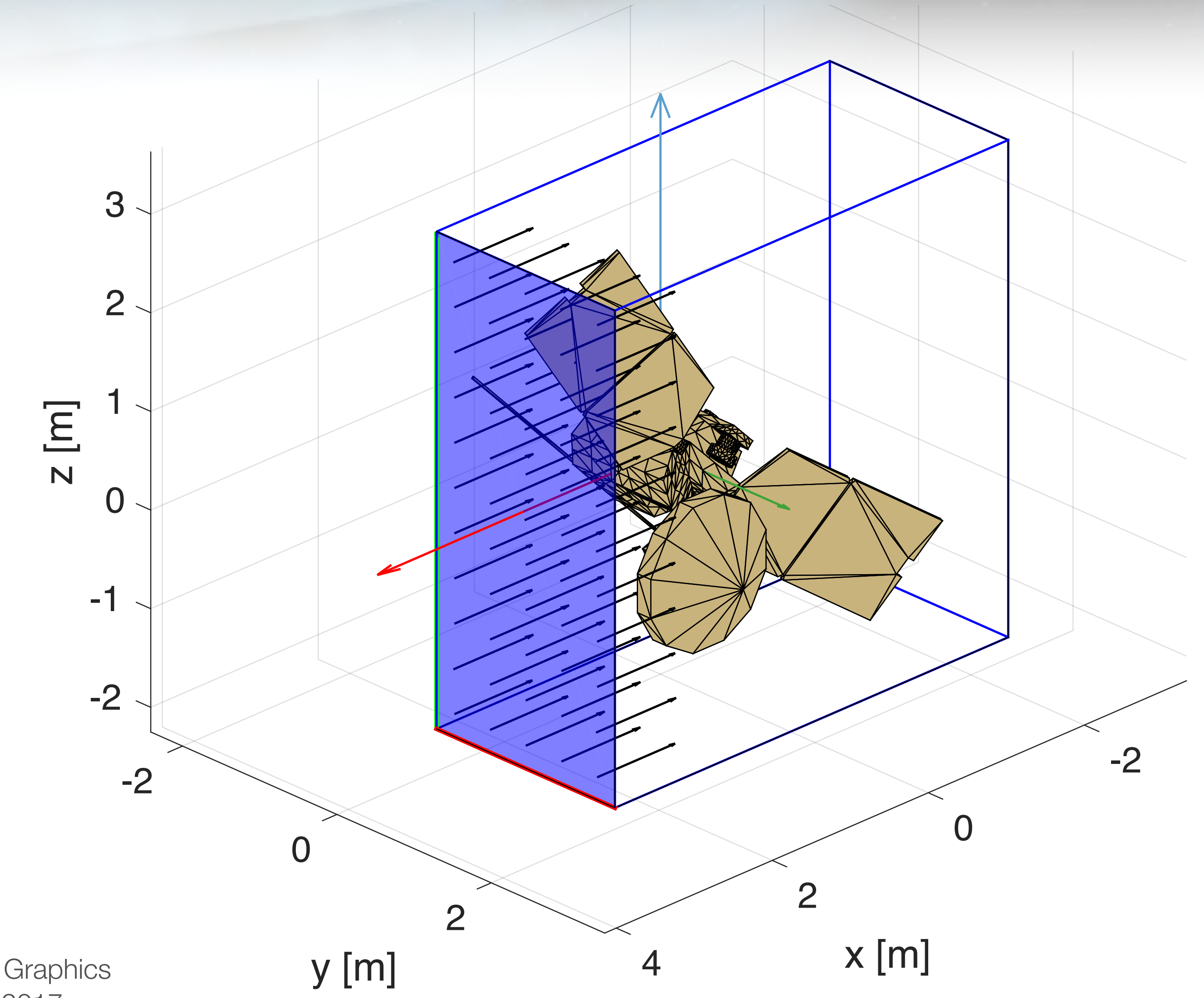
Solar Radiation Pressure Modeling





GPU Usage for Solar Radiation Pressure Modeling

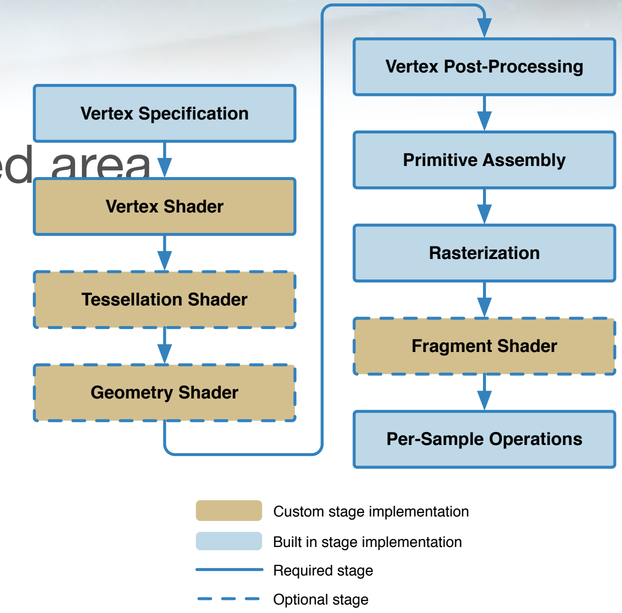
- Setup:
  - Use full CAD Model
  - Add SRP properties to surface texture
- Usage:
  - Within Basilisk this model is loaded up once onto the GPU.
  - Can handle time varying geometries through the use of articulated mesh structures
  - Each time step the orientation relative to the solar flux is adjusted based on the current spacecraft orientation
  - The forces acting on each CAD facet are added up to yield a net force and torque

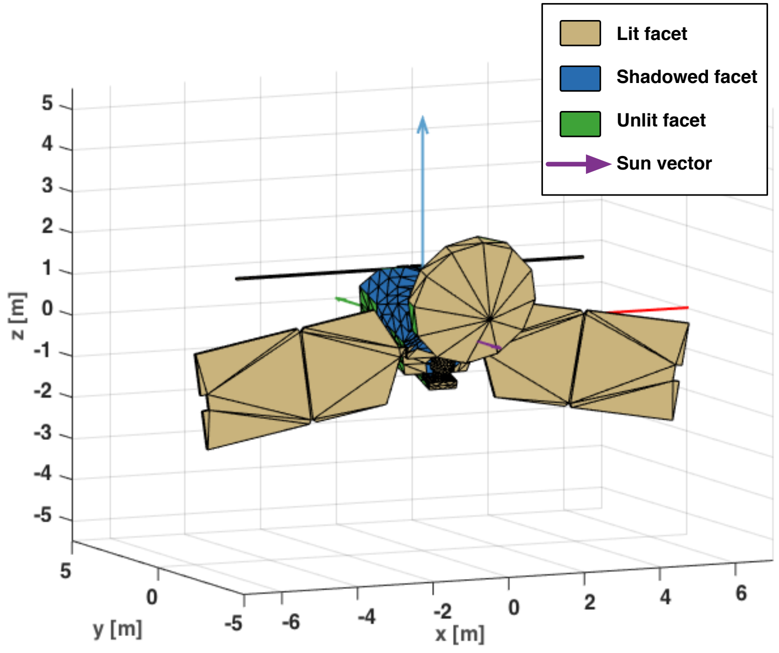


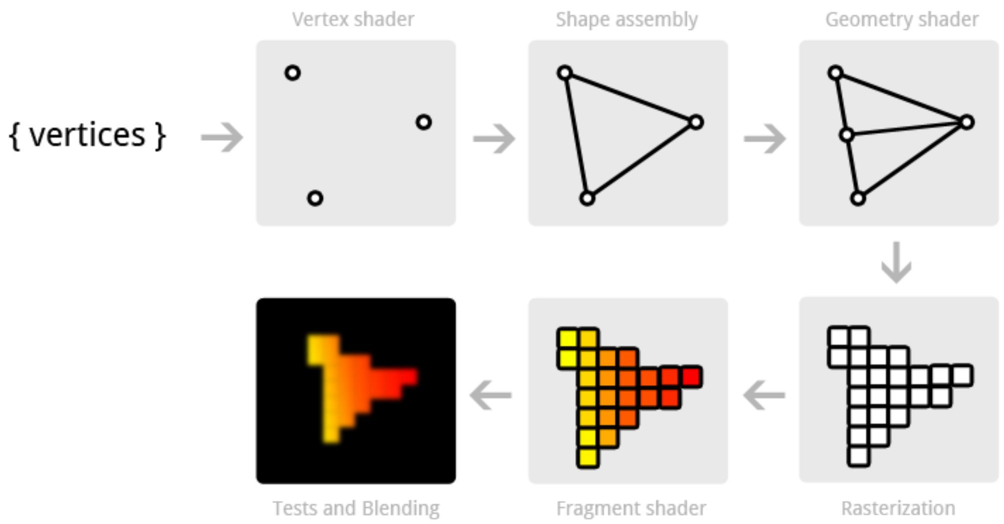
P. Kenneally and H. Schaub, “Modeling Of Solar Radiation Pressure and Self-Shadowing Using Graphics Processing Unit,” AAS Guidance, Navigation and Control Conference, Breckenridge, Feb. 2–8, 2017.

Faceted SRP Using OpenGL Render Pipeline

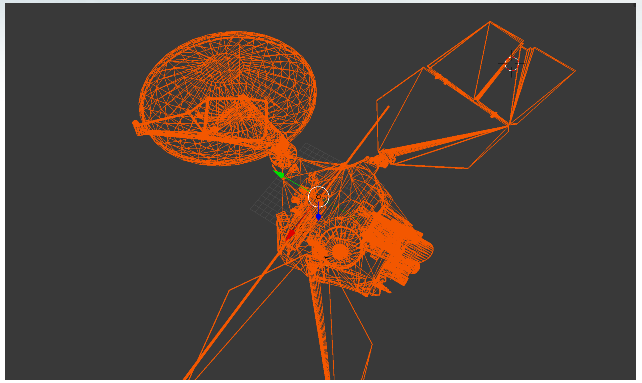
- Use built in OpenGL depth testing
- Use built in OpenGL rasterization to generate sun projected area
- OpenGL-CL shared data context



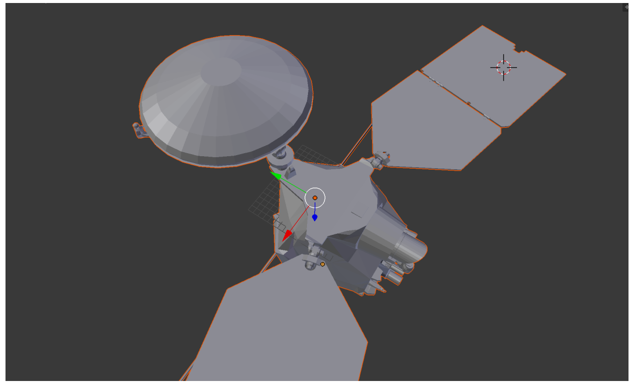




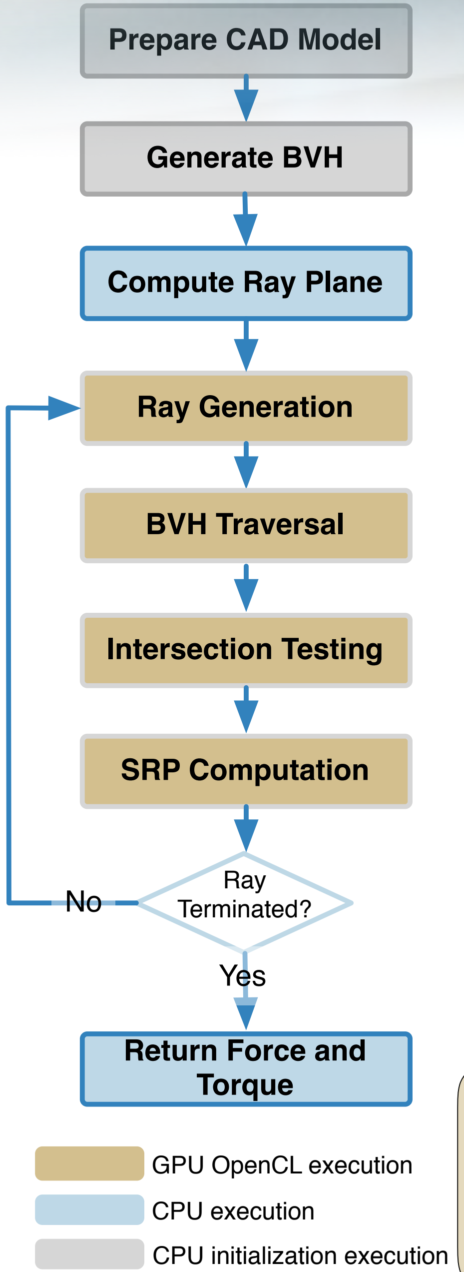
GPU Based Solar and Atmospheric Pressure Drag

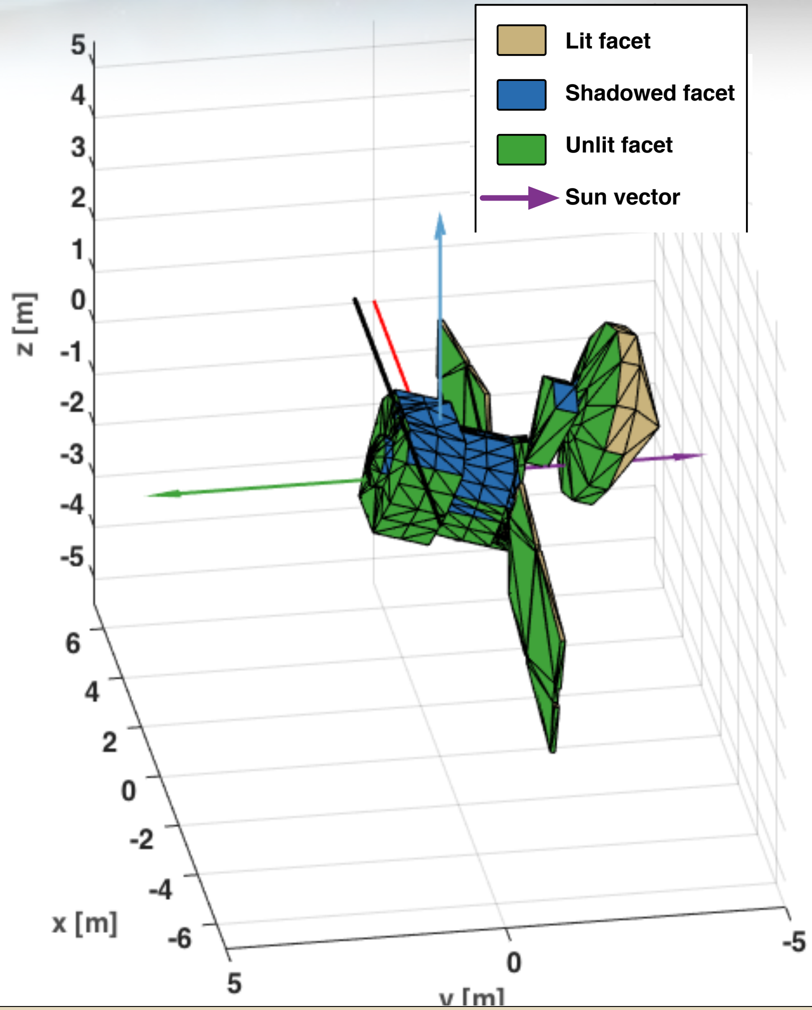


CAD model with material properties



Use GPU to evaluate forces and torques using custom vertex shaders



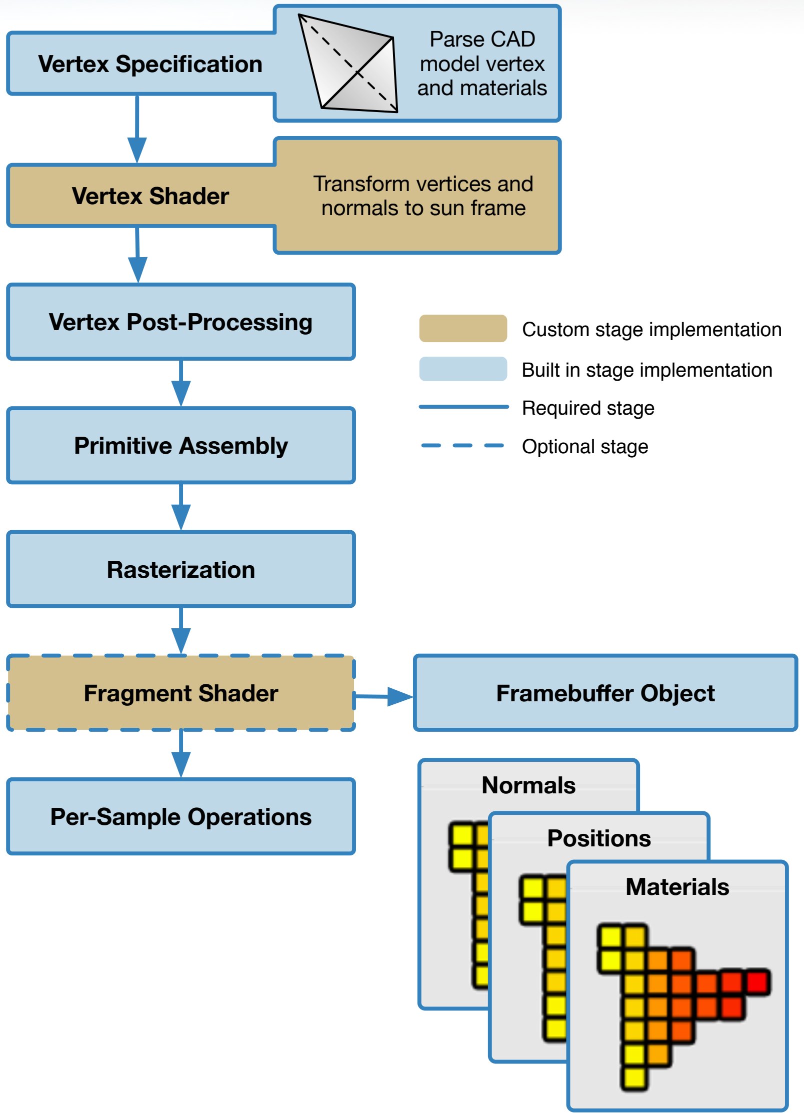


Drag forces can be rapidly computed on complex, time-varying geometries at speeds suitable for even hardware in-the-loop scenarios.

P. Kenneally and H. Schaub, “High Geometric Fidelity Modeling Of Solar Radiation Pressure Using Graphics Processing Unit,” *AAS Spaceflight Mechanics Meeting*, Napa Valley, California, February 14–18, 2016. Paper No. 16-500.

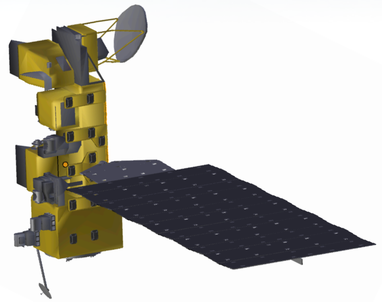
Custom OpenGL Render Pipeline

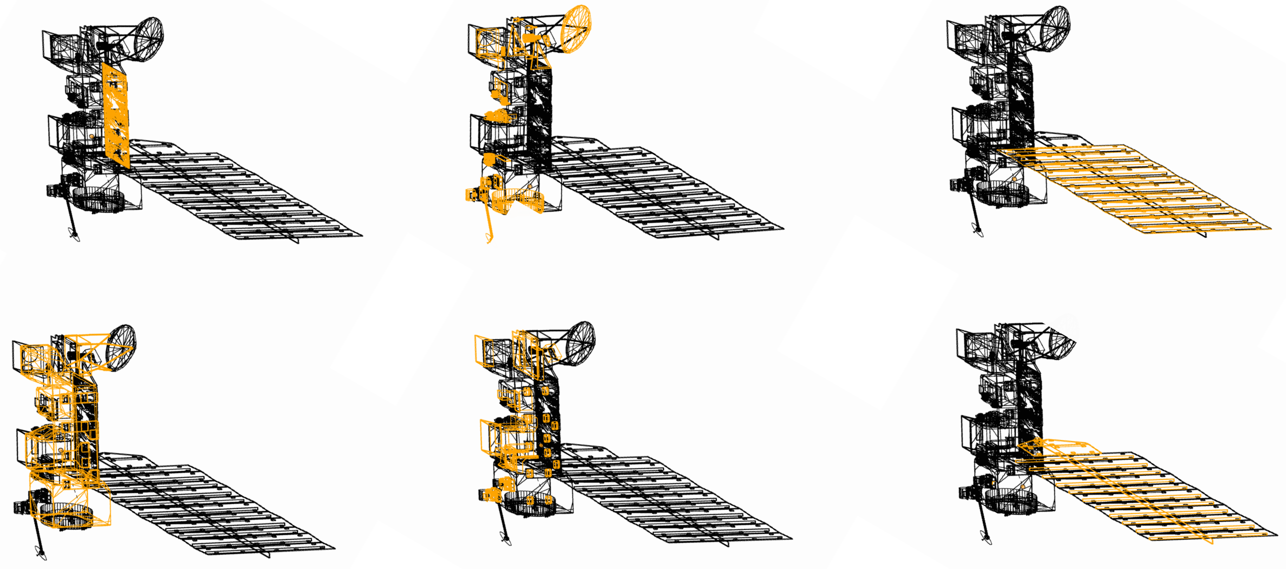
- Framebuffer Object holds the rendered output
- Textures objects (typically used for 2D image data) are attached to the Framebuffer
- Vertex Shader performs a series of frame transformations from body-frame to projection-frame
- Each vertex's position vector, normal vector and material definition is passed through
- Fragment Shader outputs sun lit model regions



Mesh and Material Definition

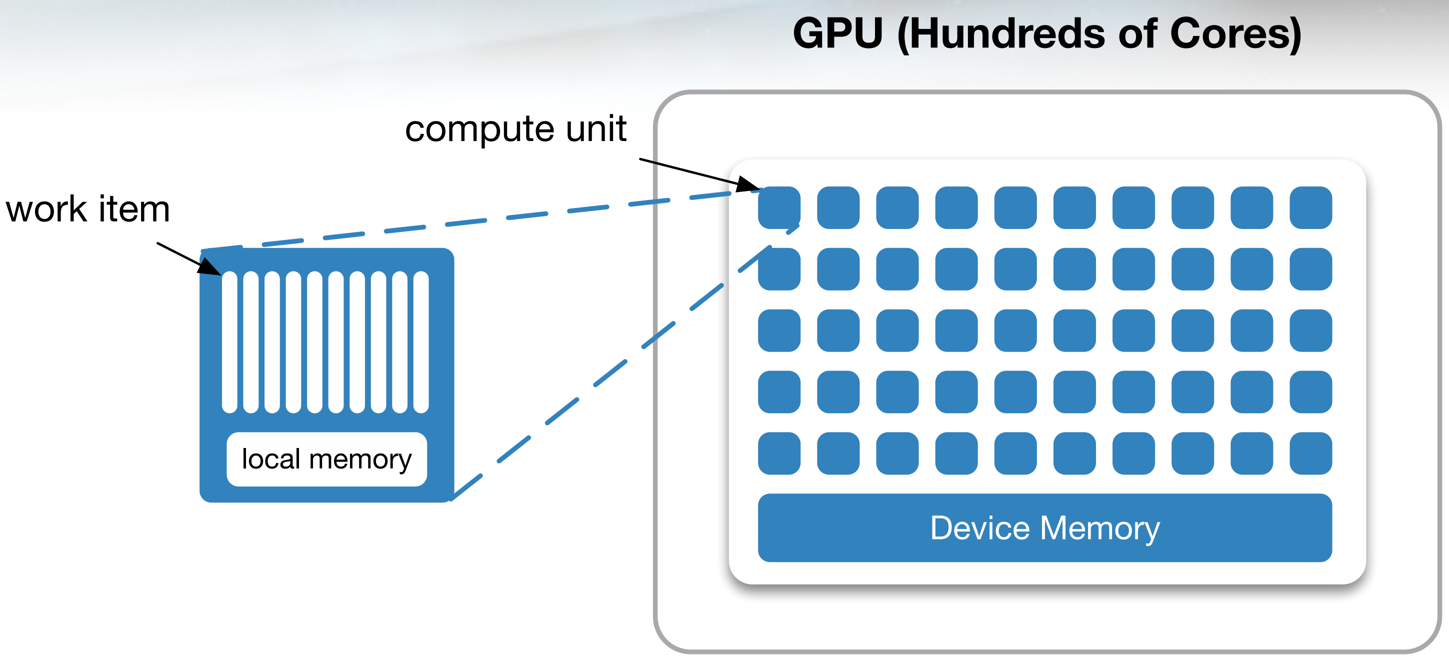
- .OBJ and .MTL files
- Each facet allocated material properties ρ_s ρ_d
- Sub-meshes defined recursively
- A mesh may have multiple sub-meshes





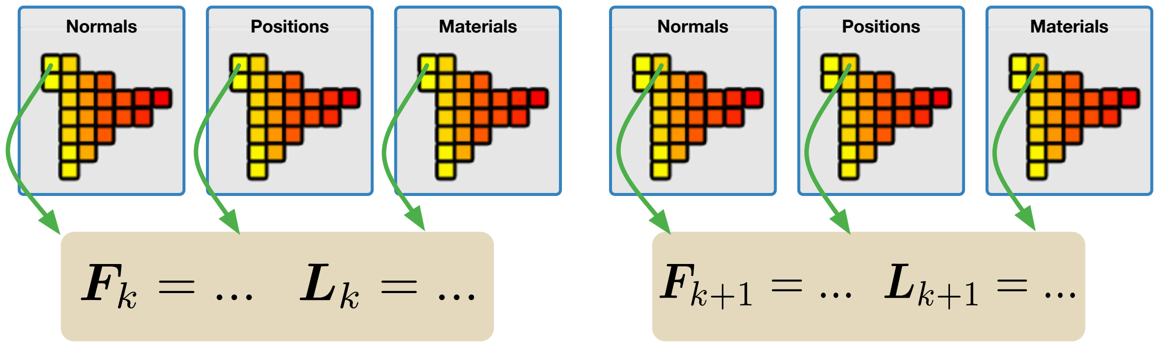
OpenCL Steps - SRP

- Work Groups (WG) = single GPU core
- Work Item (WI) = thread within WG
- Each WI will perform the SRP computation for a single pixels in the rasterized spacecraft model rendering

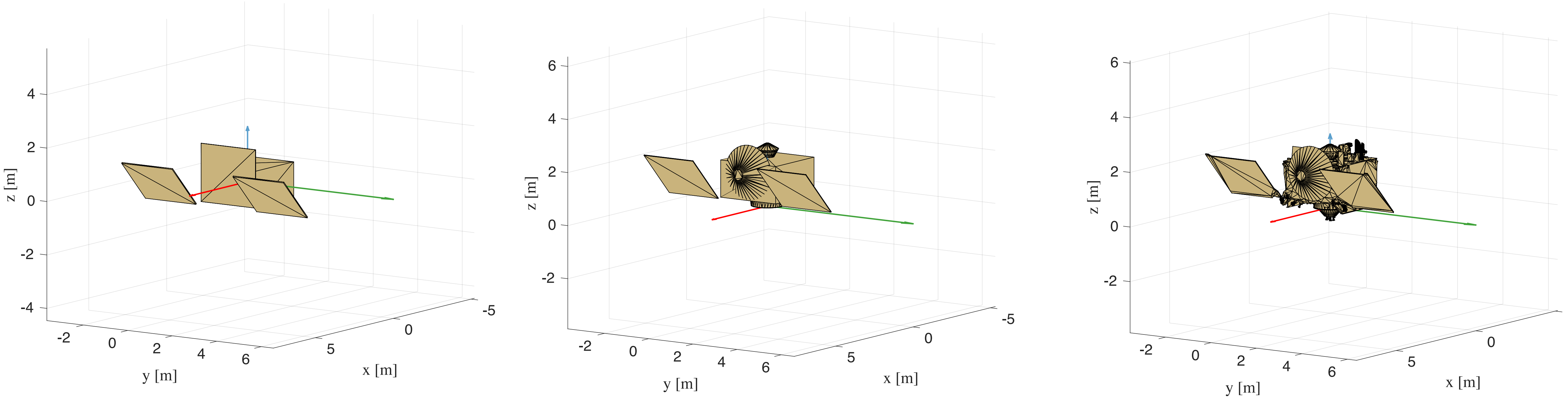


$$F_{\odot_k} = -P(|r_{\odot}|)A_k \cos(\theta_k)\left\{(1-\rho_{s_k})\hat{s} + \left[\frac{2}{3}\rho_{d_k} + 2\rho_{s_k}\cos(\theta_k)\right]\hat{n}_k\right\}$$

$$L_{\odot_k} = r_{P/C} \times F_{\odot_k}$$



Mesh Detail Impact



(a) Box and wings.

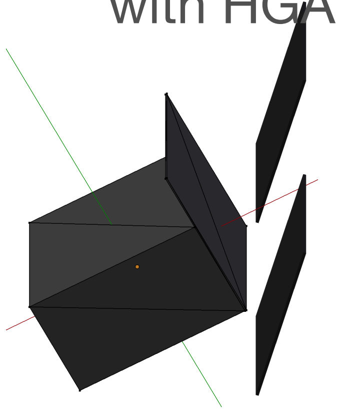
(b) High gain antenna, thruster ring and sample return module.

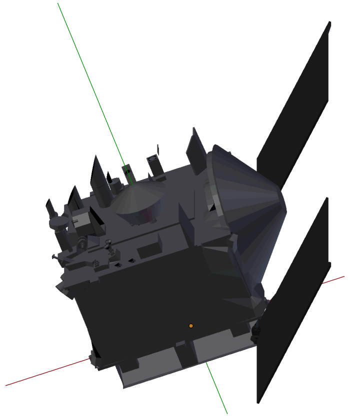
(c) Hifidelity model.

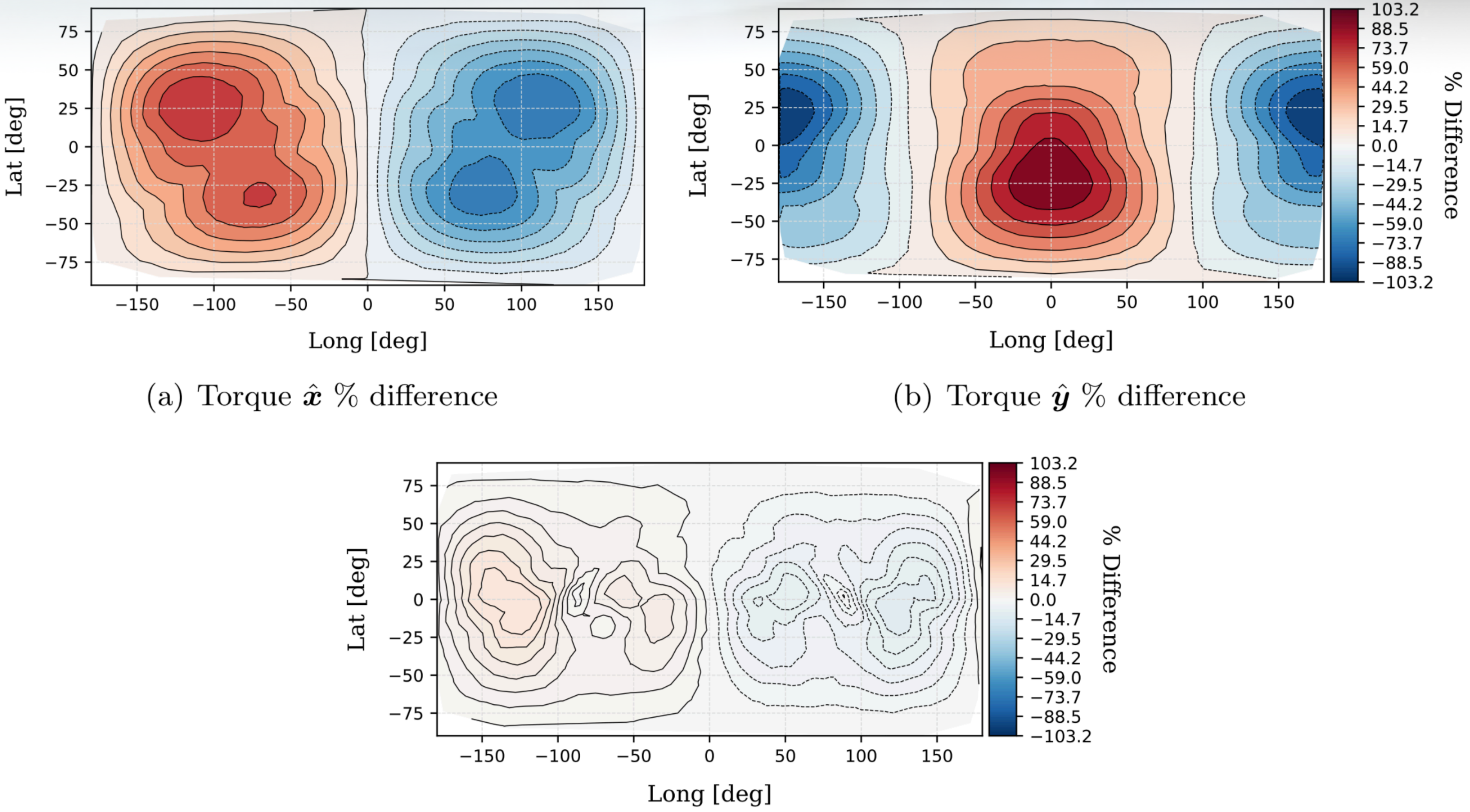
Patrick Kenneally, ``Faster than Real-Time GPGPU Radiation Pressure Modeling Methods,'' Ph.D. Dissertation, Aerospace Engineering Sciences Department, University of Colorado, Boulder, CO, May 2019.

Torque Box and Wing - Hifi

- Speed means no need t sacrifice resolution with lesser models. So we ta a look at what the sacrif would be if we used oth methods to compute bo and wing, and flat plate with HGA







(a) Torque $\hat{\boldsymbol{x}}$ % difference

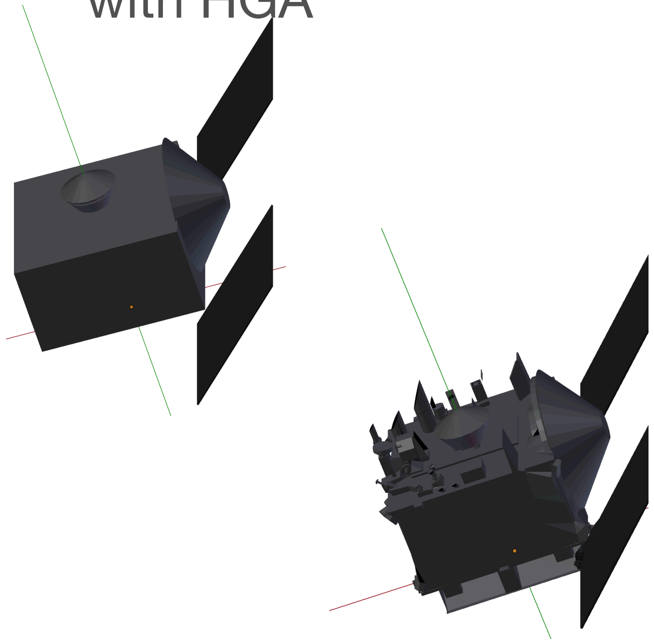
(b) Torque $\hat{\boldsymbol{y}}$ % difference

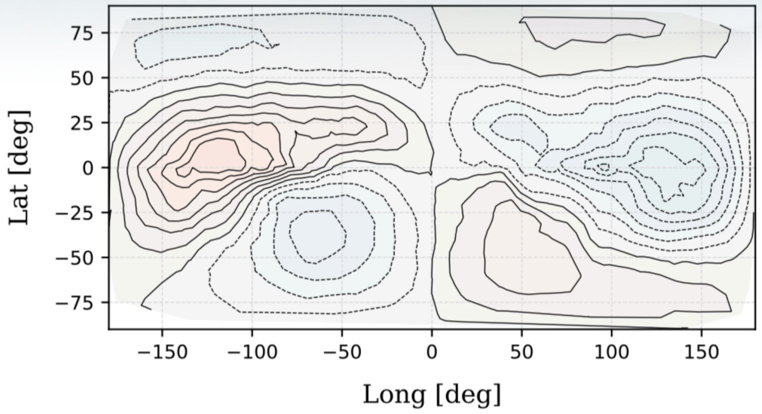
(c) Torque $\hat{\boldsymbol{z}}$ % difference

Figure 3.17: Torque percentage difference between box and wing model relative to the hifidelity model with baseline value 6.57817×10^{-5} [Nm].

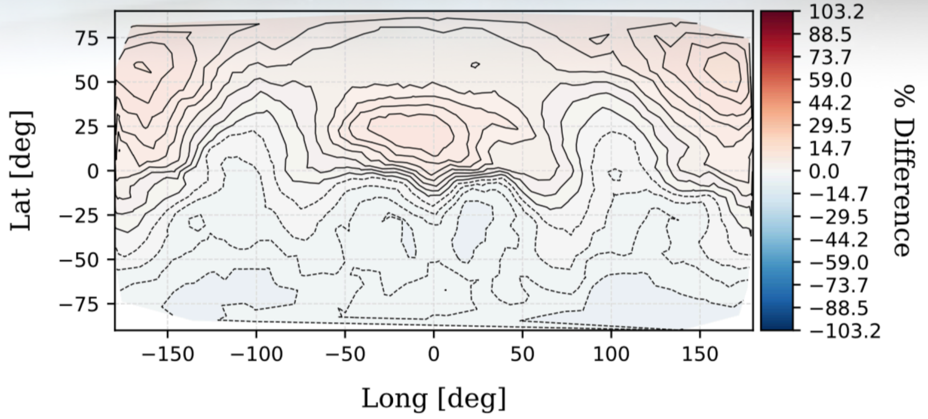
Torque HGA - Hifi

- Speed means no need t
sacrifice resolution with
lesser models. So we ta
a look at what the sacrif
would be if we used oth
methods to compute bo
and wing, and flat plate
with HGA

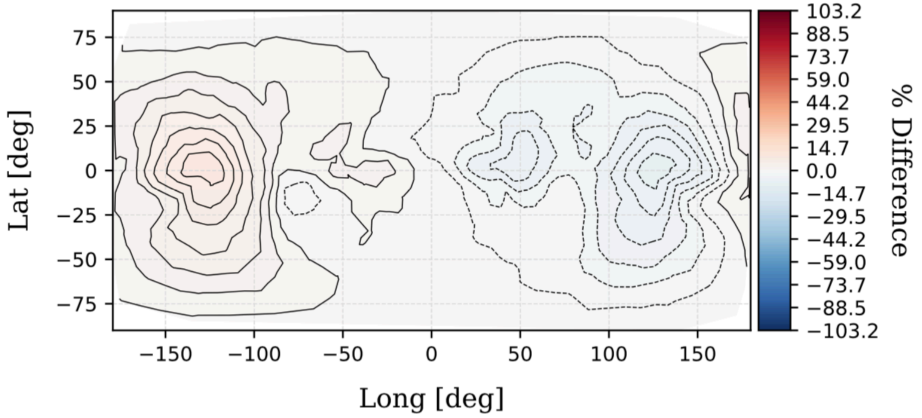




(a) Torque $\hat{\boldsymbol{x}}$ % difference



(b) Torque $\hat{\boldsymbol{y}}$ % difference

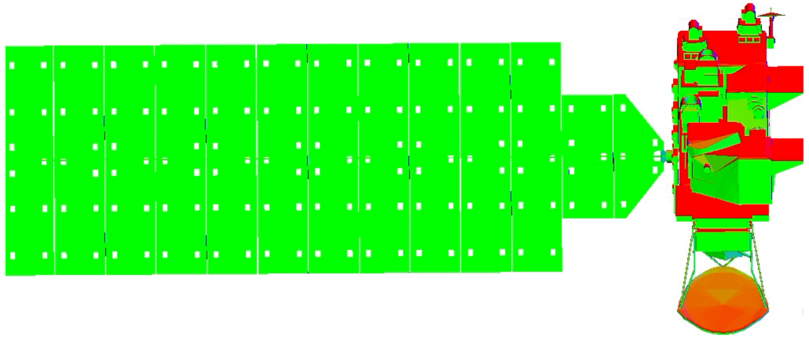


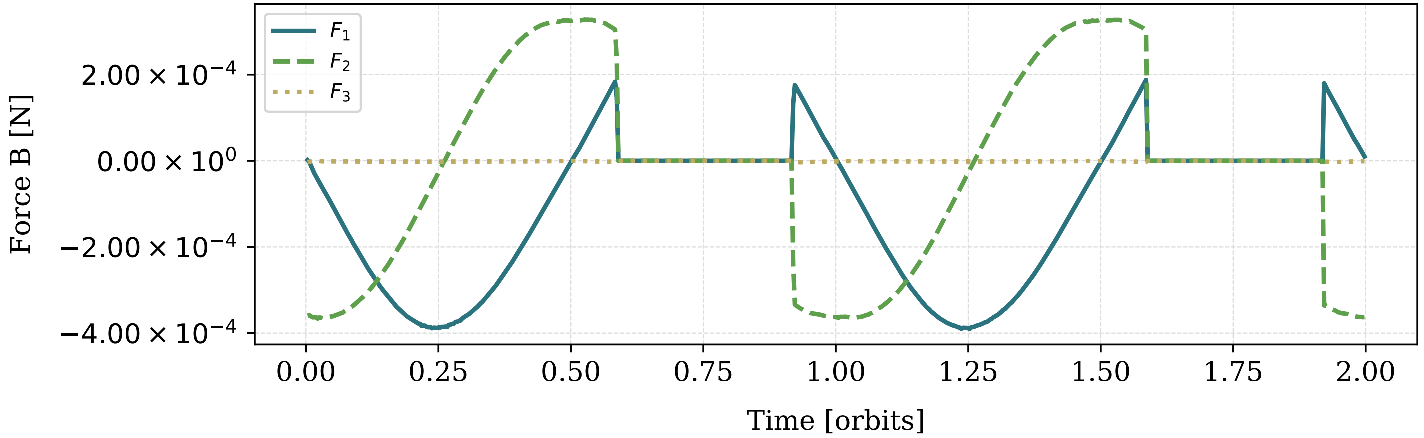
(c) Torque $\hat{\boldsymbol{z}}$ % difference

Figure 3.19: Torque percentage difference between HGA model relative to the high-fidelity model with baseline value 6.57817×10^{-5} [Nm].

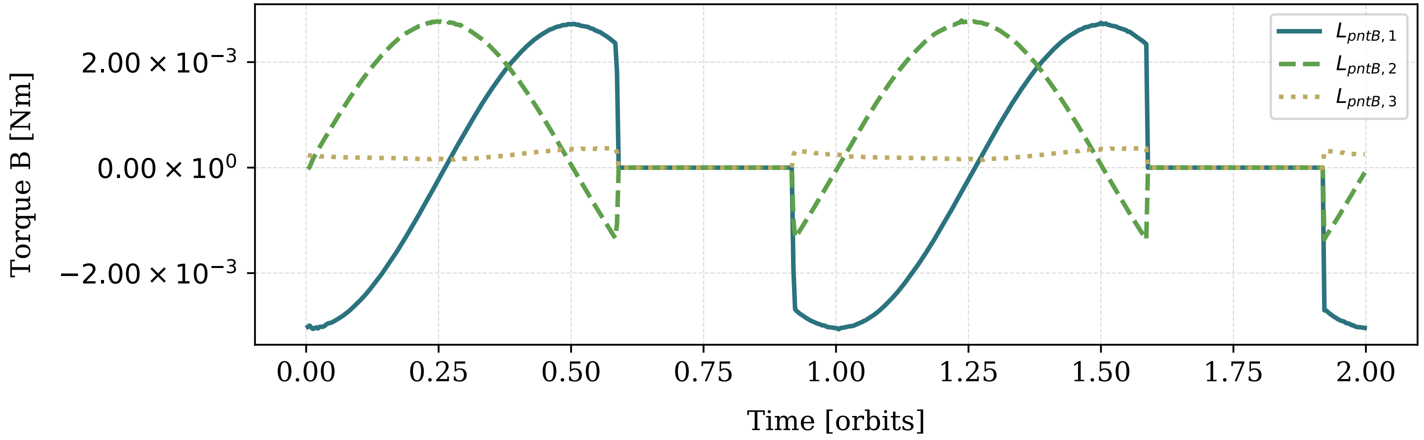
Online Simulation

- Speed means no need to sacrifice resolution with lesser models. So we take a look at what the sacrifice would be if we used other methods to compute box and wing, and flat plate with HGA



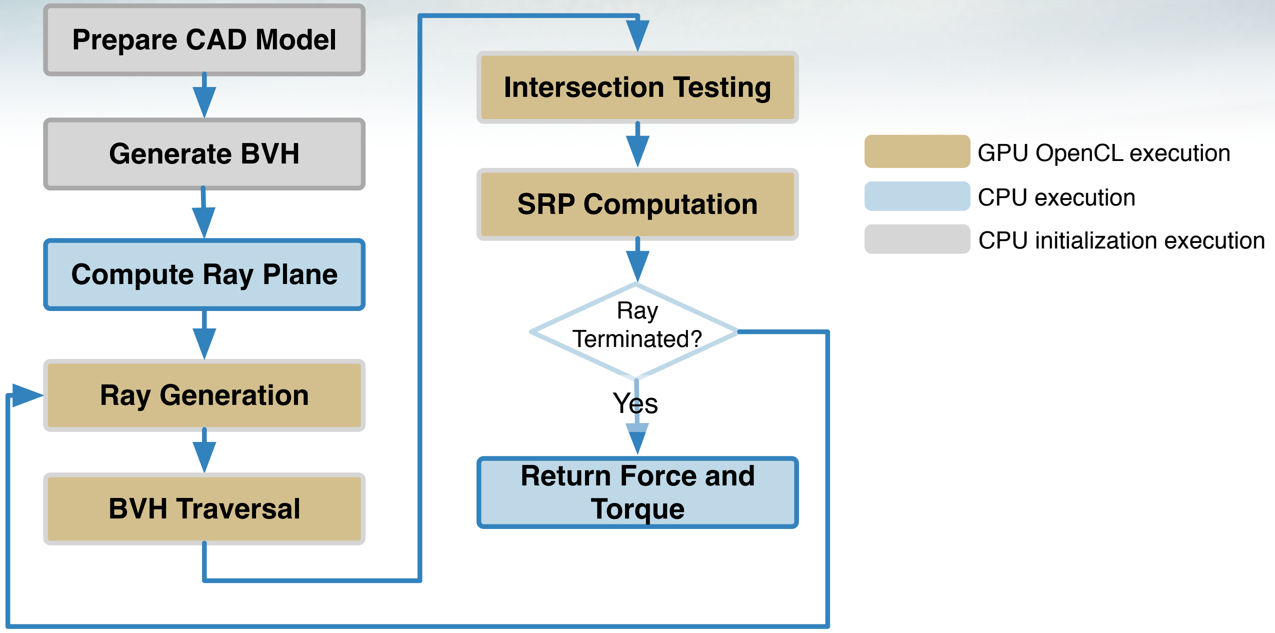


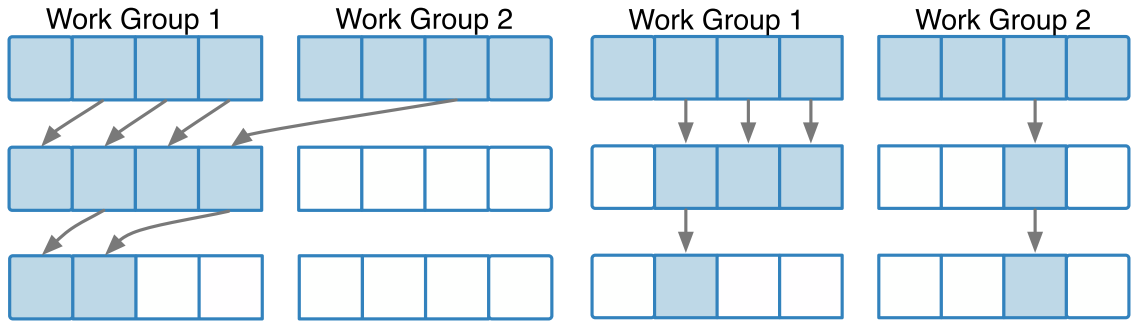
a km	7378
e	0
i, deg	90
M_0, deg	90
Ω, deg	0
ω, deg	0



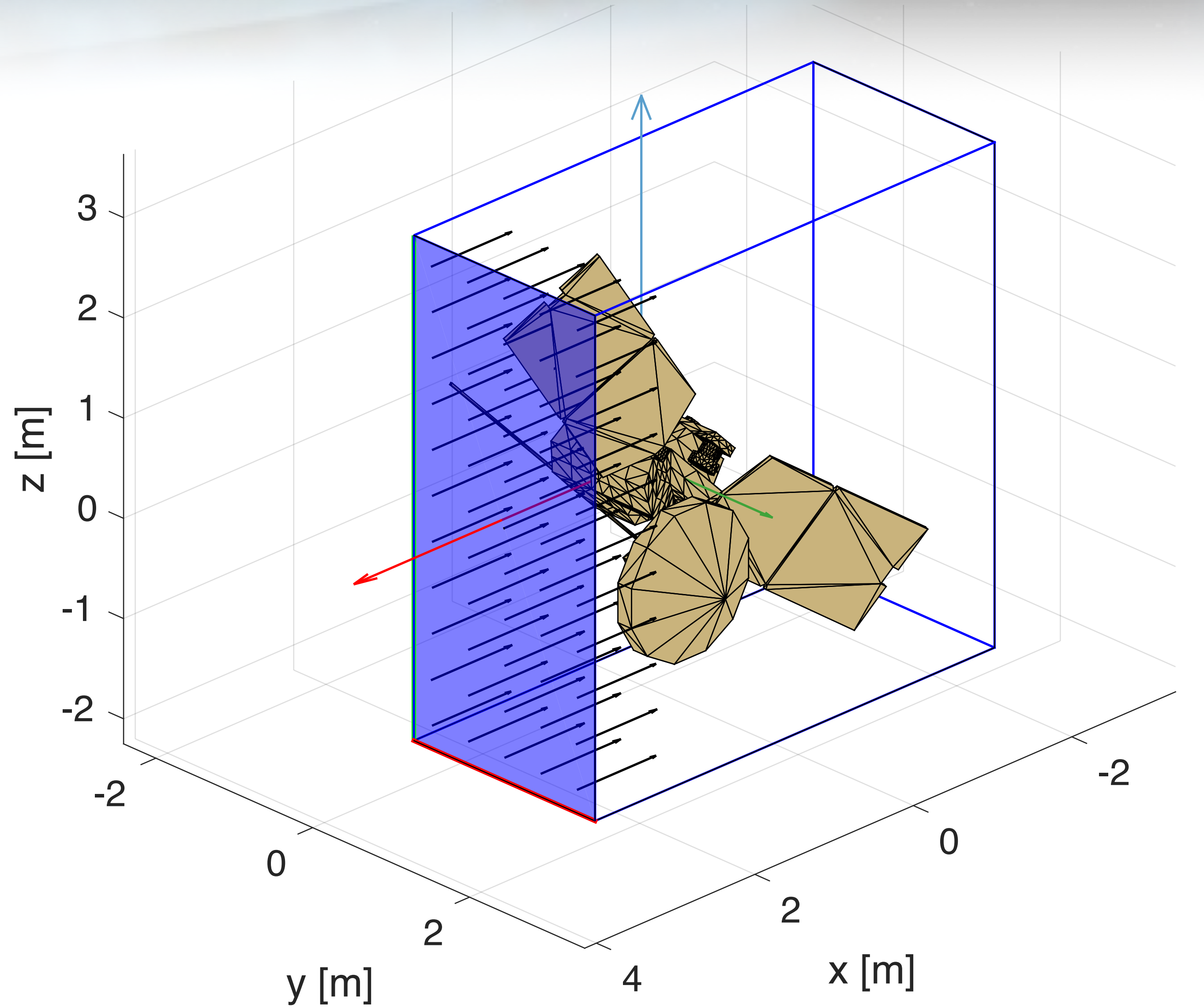
Material	Specular (ρ_s)	Diffuse (ρ_d)
Gold MLI	0.184	0.736
Silver MLI	0.66	0.16
Germanium MLI	0.3	0.3
Solar array rear	0.1	0.3
Solar array front	0.023	0.092
Solar array boom	0.3	0.3

Ray Tracing SRP Modeling





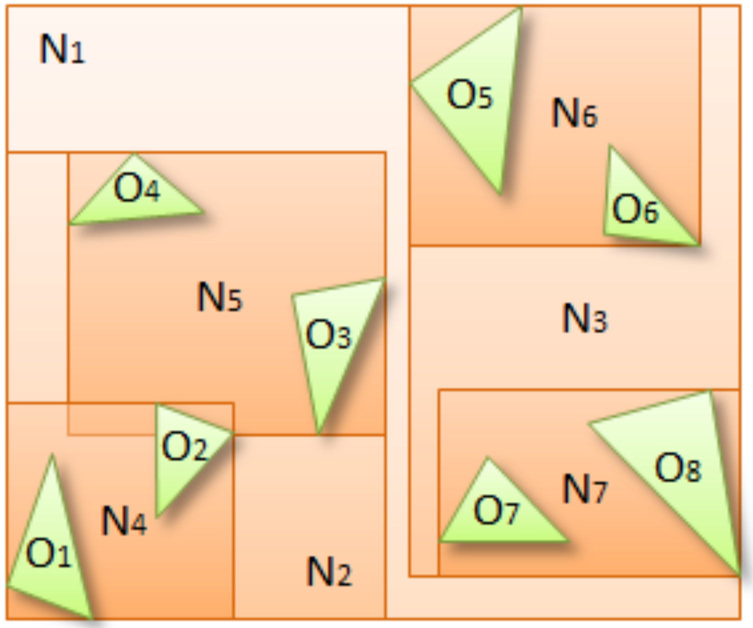
(a) SIMD by ray (b) SIMD by pixel

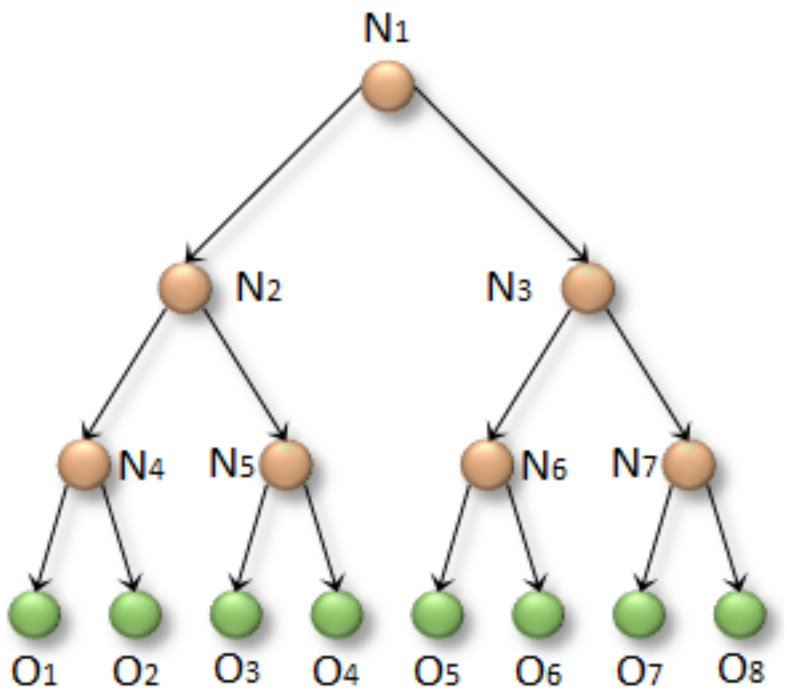


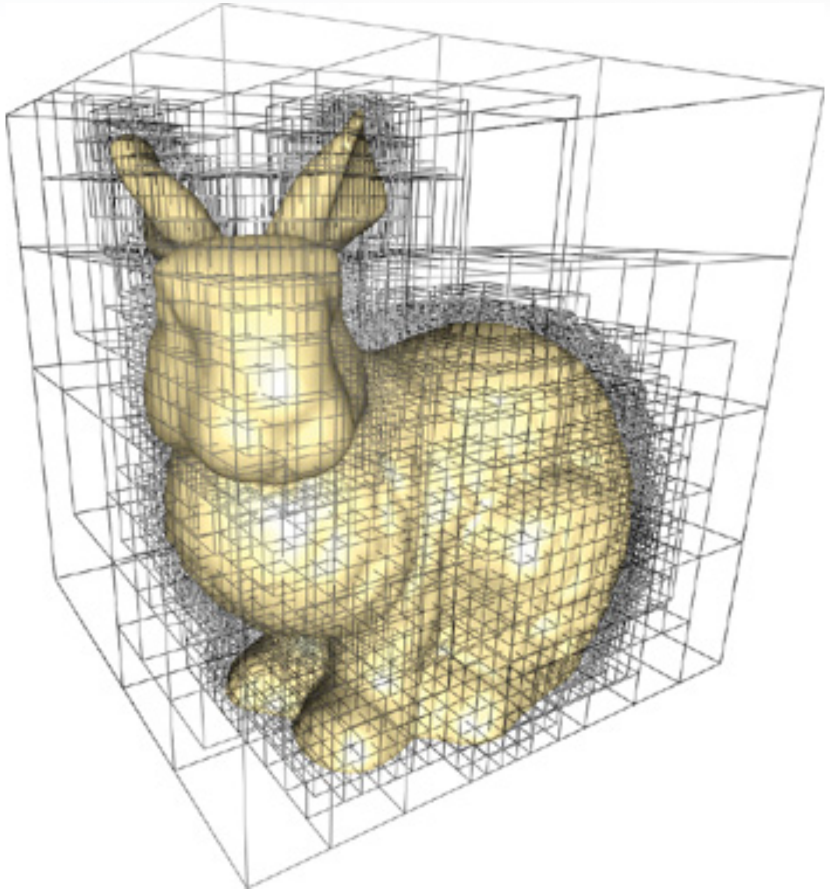
P. Kenneally and H. Schaub, "Fast Spacecraft Solar Radiation Pressure Modeling By Ray-Tracing On Graphic Processing Unit," AAS Guidance and Control Conference, Breckenridge, CO, February 2–7, 2018.

Bounding Volume Hierarchy

- ***BVH Goal***: reduce the ray-object intersection search space

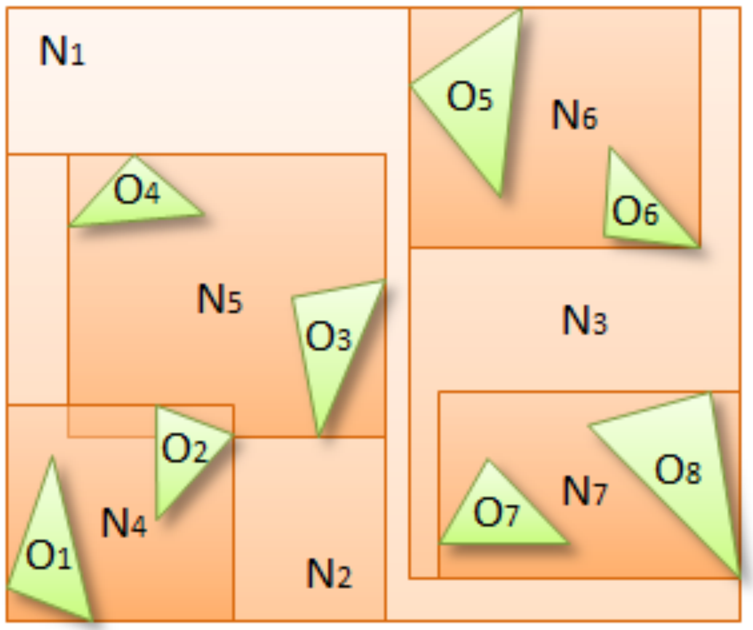


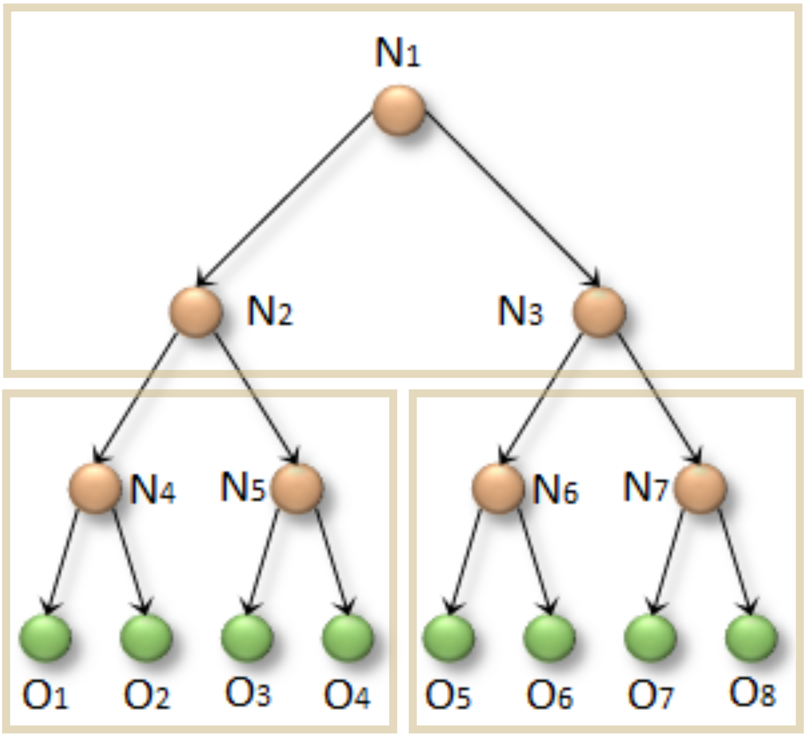




Bounding Volume Hierarchy

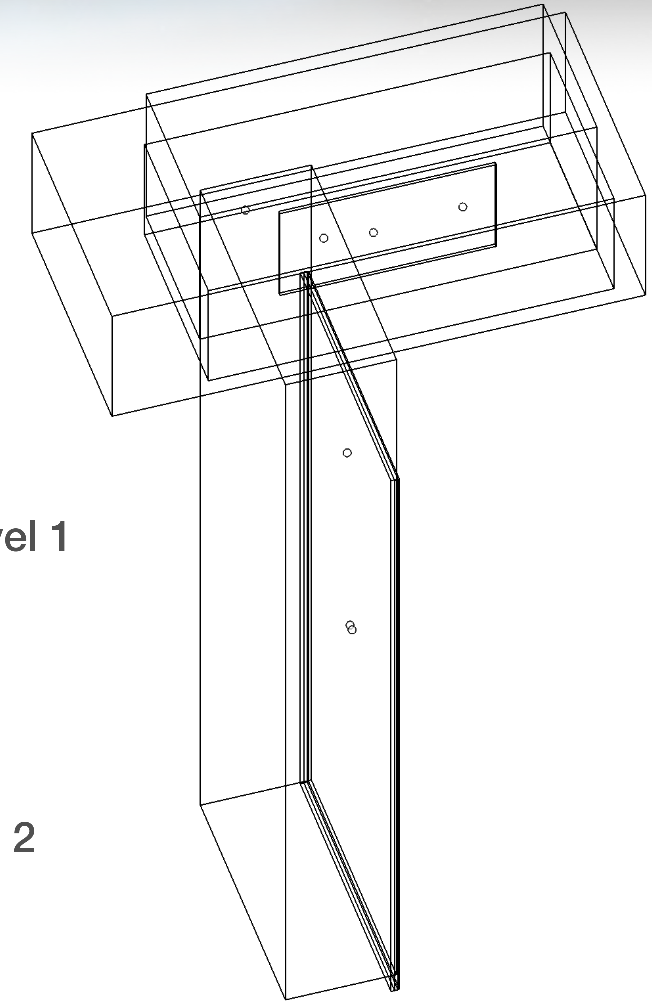
- ***BVH Goal***: reduce the ray-object intersection search space
- Two level BVH prevents rebuilding BVH when mesh articulates



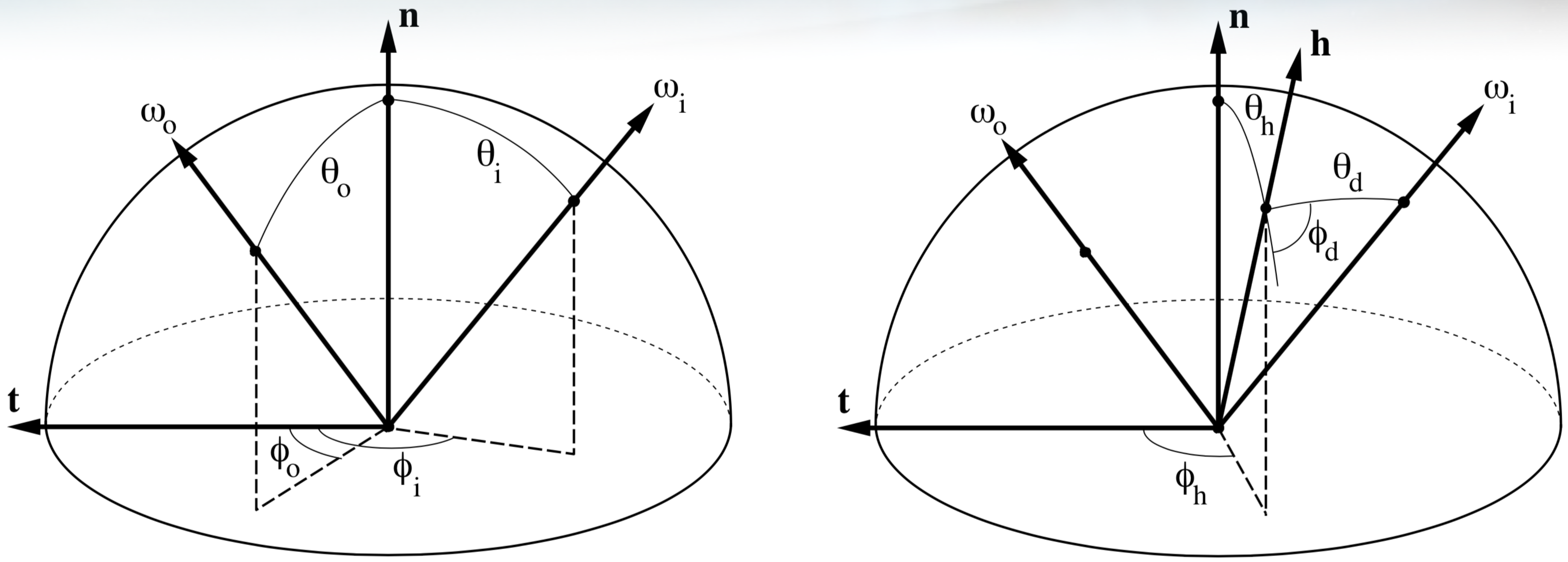


BVH Level 1

BVH Level 2



Bidirectional Reflection Distribution Functions



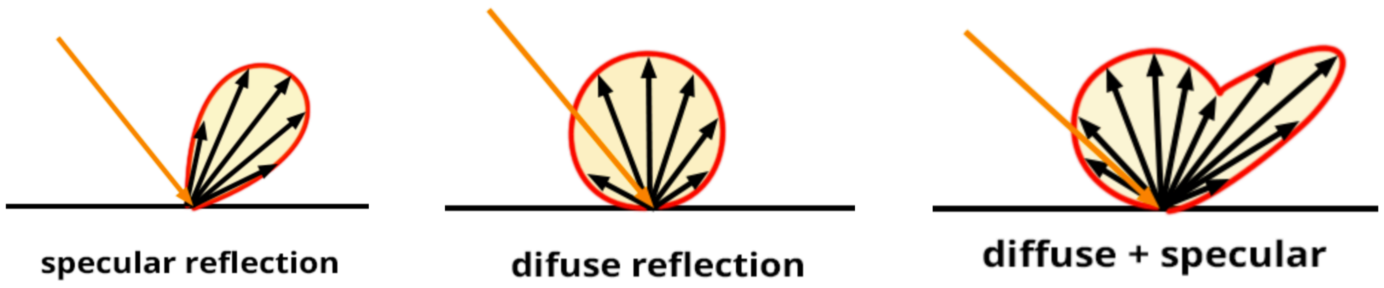
(a) BRDF is parameterized by the intuitive (θ_i, ϕ_i) and (θ_o, ϕ_o)

(b) BRDF is parameterized as function of the half angle (θ_h, ϕ_h) and a difference angle (θ_d, ϕ_d)

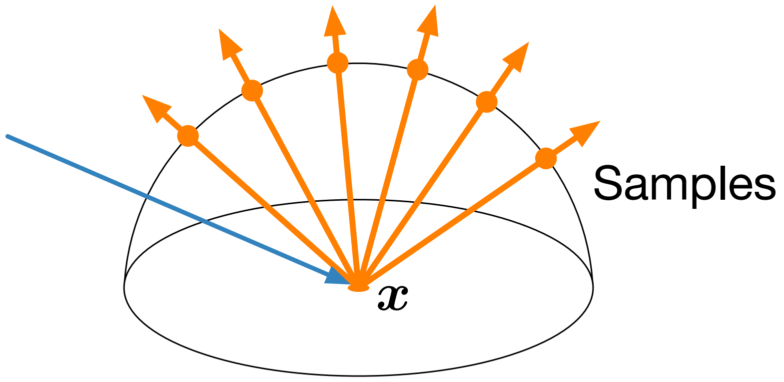
P. Kenneally and H. Schaub, “Spacecraft Radiation Pressure Using Complex Bidirectional-Reflectance Distribution Functions On Graphics Processing Unit,” AAS Spaceflight Mechanics Meeting, Maui, Hawaii January 13–17, 2019.

Monte Carlo Evaluation of BRDFs

- Evaluation of anisotropic BRDFs can be slow using quadrature

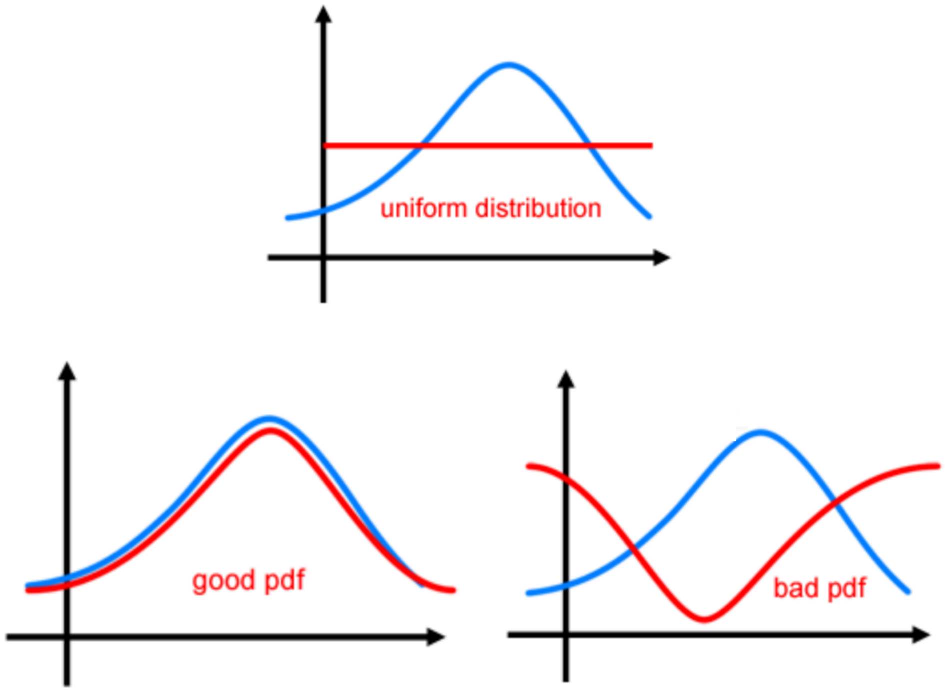


$$L_o(w_o) = \int_{H^2} L(w_i) f_s(w_i \rightarrow w_o) d\sigma^{\perp}(w_i)$$



Monte Carlo Estimator

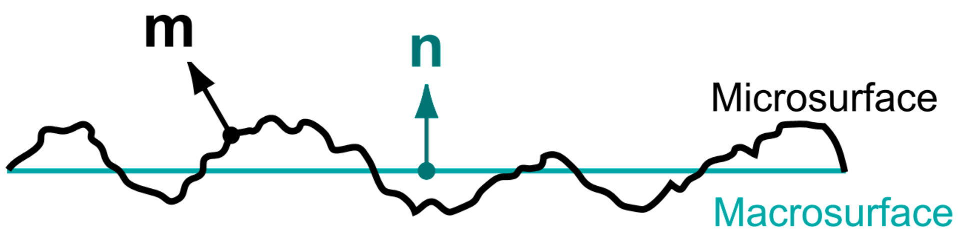
$$\langle F^N \rangle = \frac{1}{N} \sum_{i=0}^{N-1} \frac{f(X_i)}{pdf(X_i)}$$



Microfacet BRDF

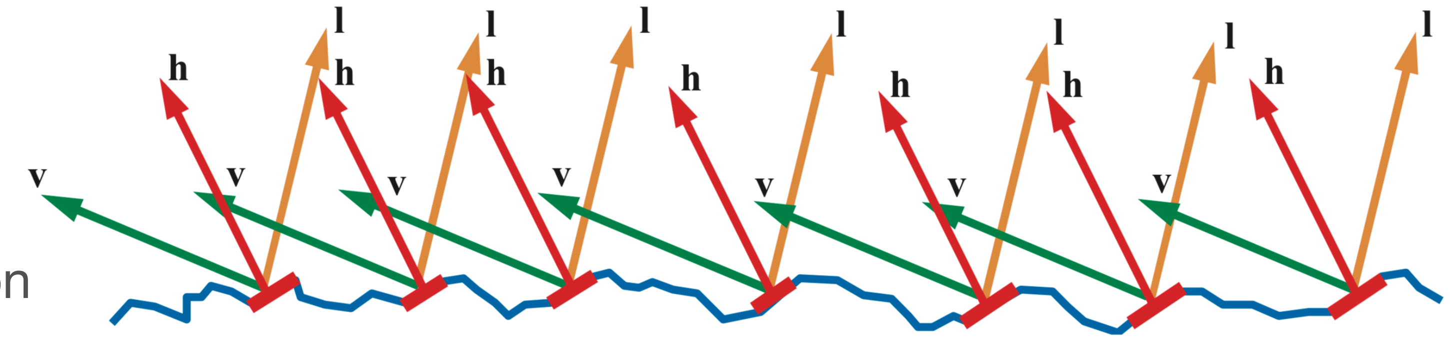
- Surface points with normal m = h are oriented such that they reflect l into v

- Other surface points do not contribute to the BRDF.



General micro-facet model

$$R_s = \frac{D(\boldsymbol{\omega}_h)G(\boldsymbol{\omega}_o, \boldsymbol{\omega}_i)F(\boldsymbol{\omega}_o)}{4(\hat{\boldsymbol{n}} \cdot \hat{\boldsymbol{w}}_o)(\hat{\boldsymbol{n}} \cdot \hat{\boldsymbol{\omega}}_i)}$$



Hoffman. N, « Physically Based Shading Math and Notes »

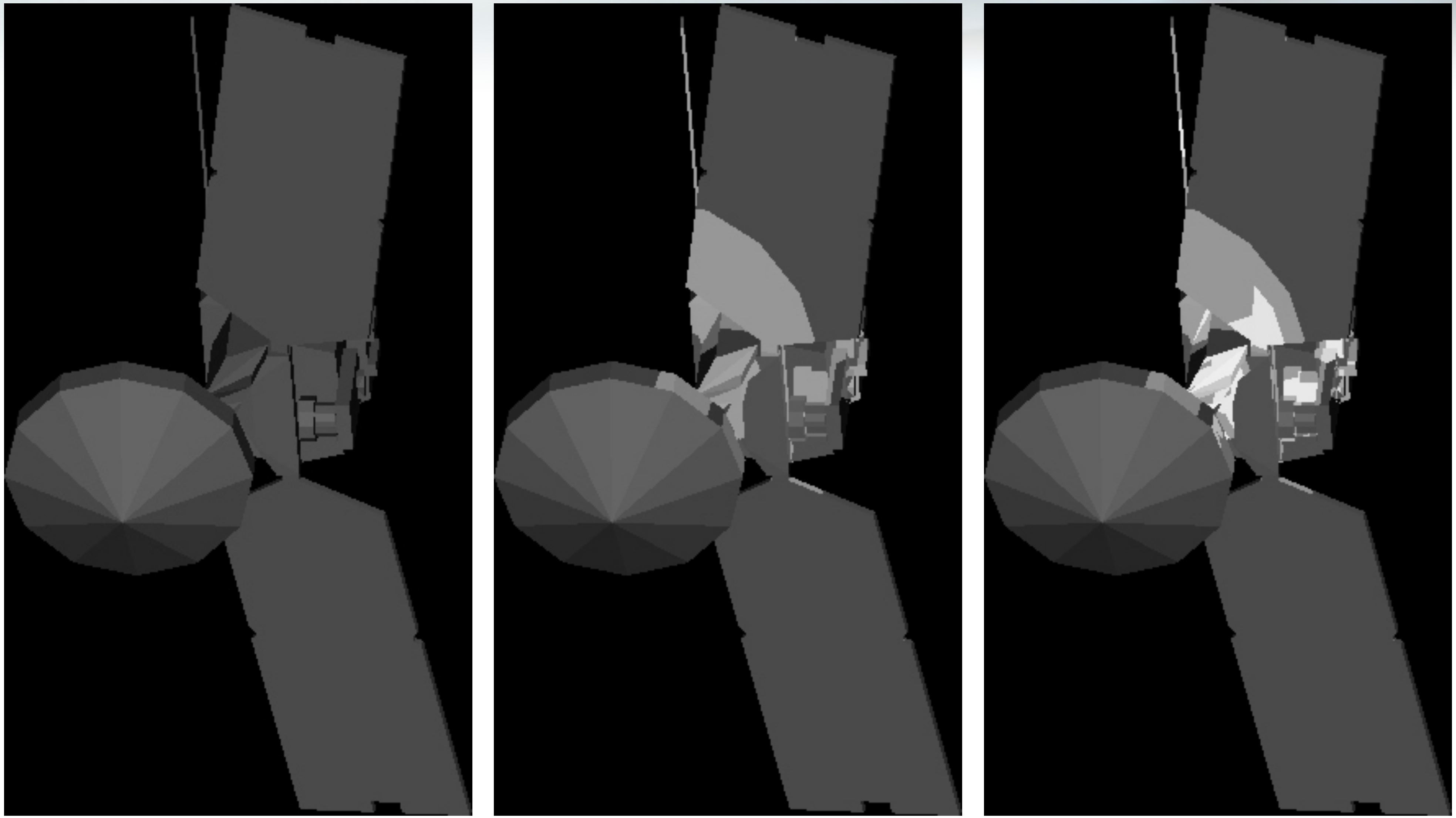
- *D* is the microgeometry normal distribution function

- *G* is the geometry function

- *F* is the Fresnel reflection factor

Ray Bounce Illustration

P. Kenneally and H. Schaub, “Modeling Of Solar Radiation Pressure and Self-Shadowing Using Graphics Processing Unit,” *AAS Guidance, Navigation and Control Conference*, Breckenridge, Feb. 2–8, 2017.

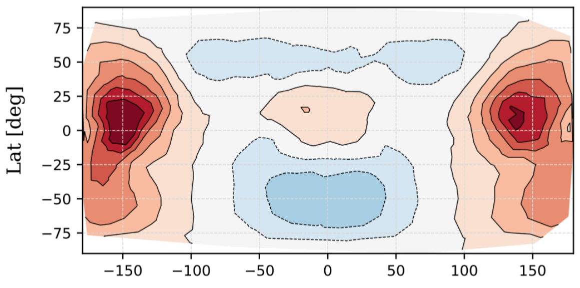


(a) One bounce (b) Two bounces (c) Three bounces

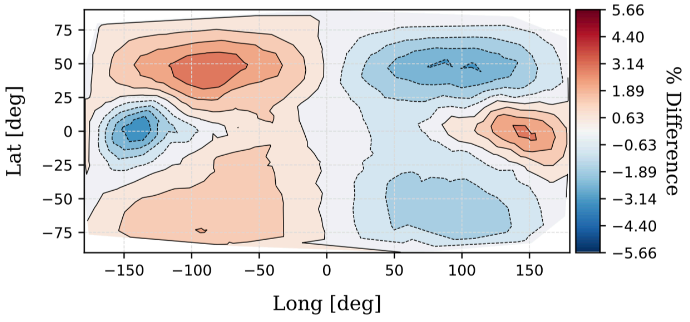
Multiple Bounce High-Fidelity Model

$$F_{\mathrm{i+1,\,i}} = \frac{|\boldsymbol{F}_{\mathrm{i+1}}| - |\boldsymbol{F}_{\mathrm{i}}|}{F_{\mathrm{base}}} \times 100 \qquad F_{\mathrm{base}} = \frac{1}{N}\sum_{n=1}^{N} |\boldsymbol{F}_{\mathrm{n}}|$$

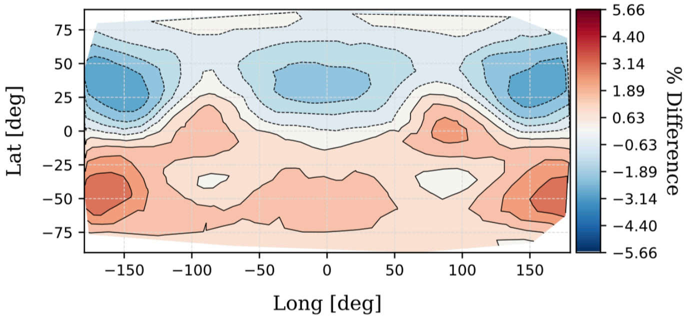
5.62995 x 10^{-5} [N]



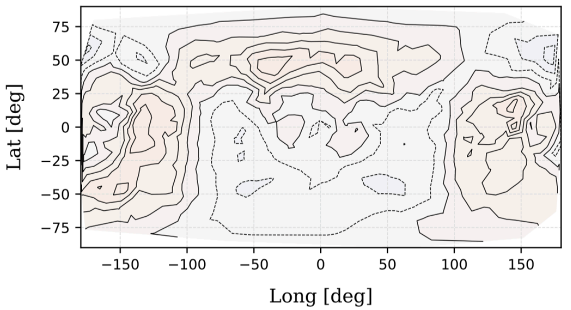
(a) Force x % difference intersection 2 - 1



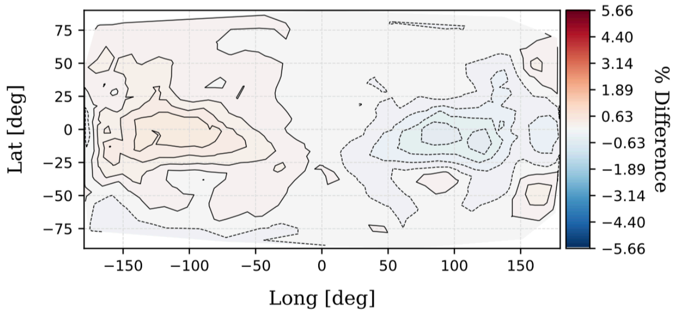
(b) Force y % difference intersection 2 - 1



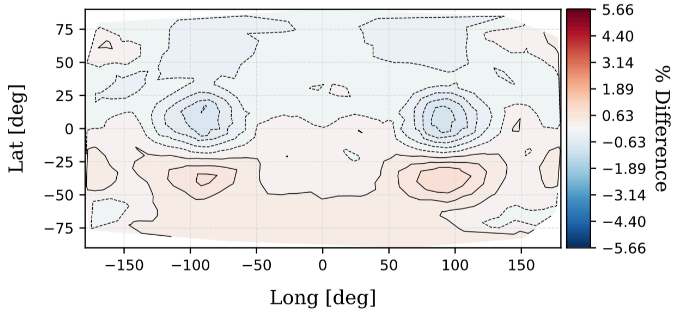
(c) Force z % difference intersection 2 - 1



(a) Force x % difference intersection 3 - 2



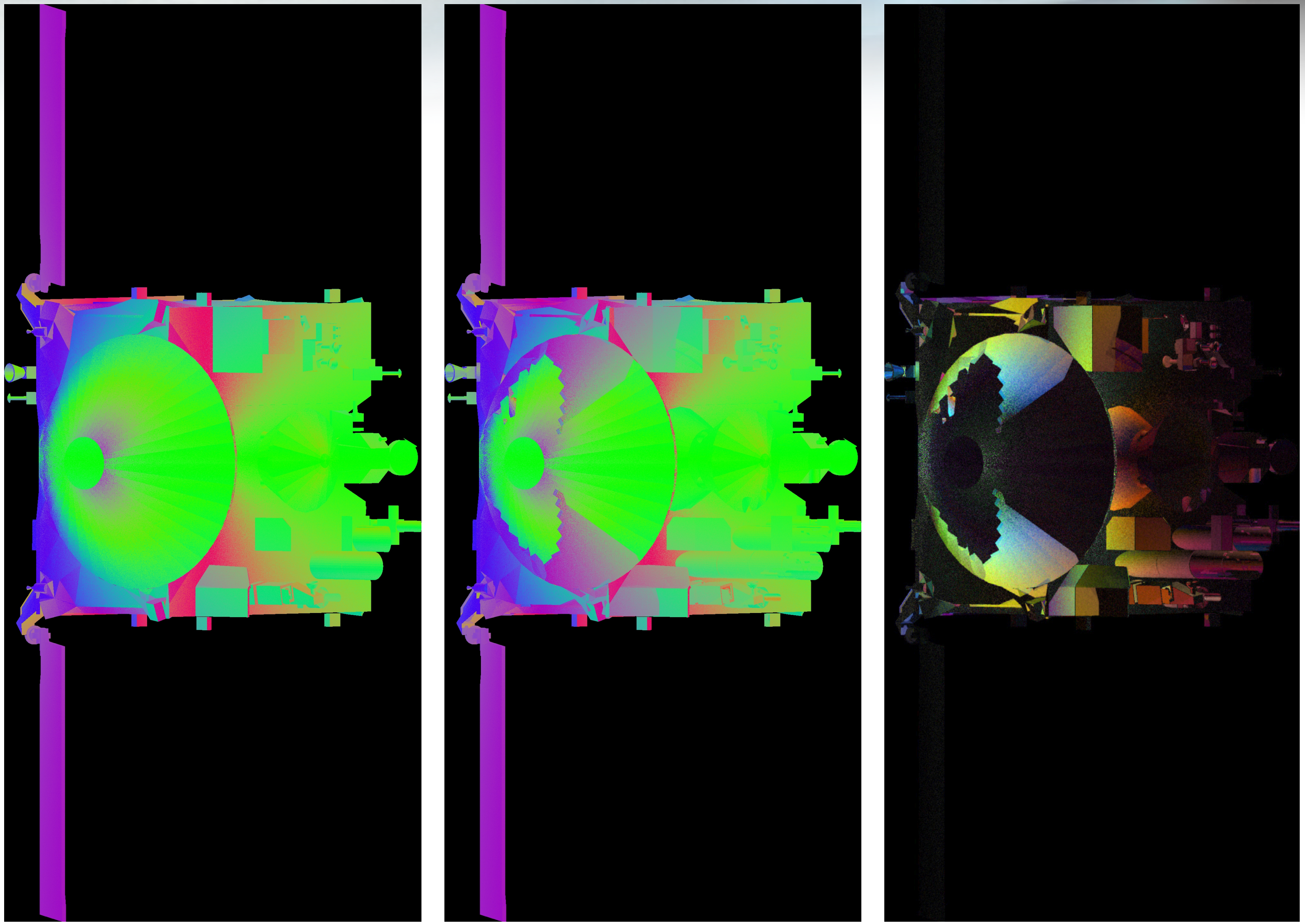
(b) Force y % difference intersection 3 - 2



(c) Force z % difference intersection 3 - 2

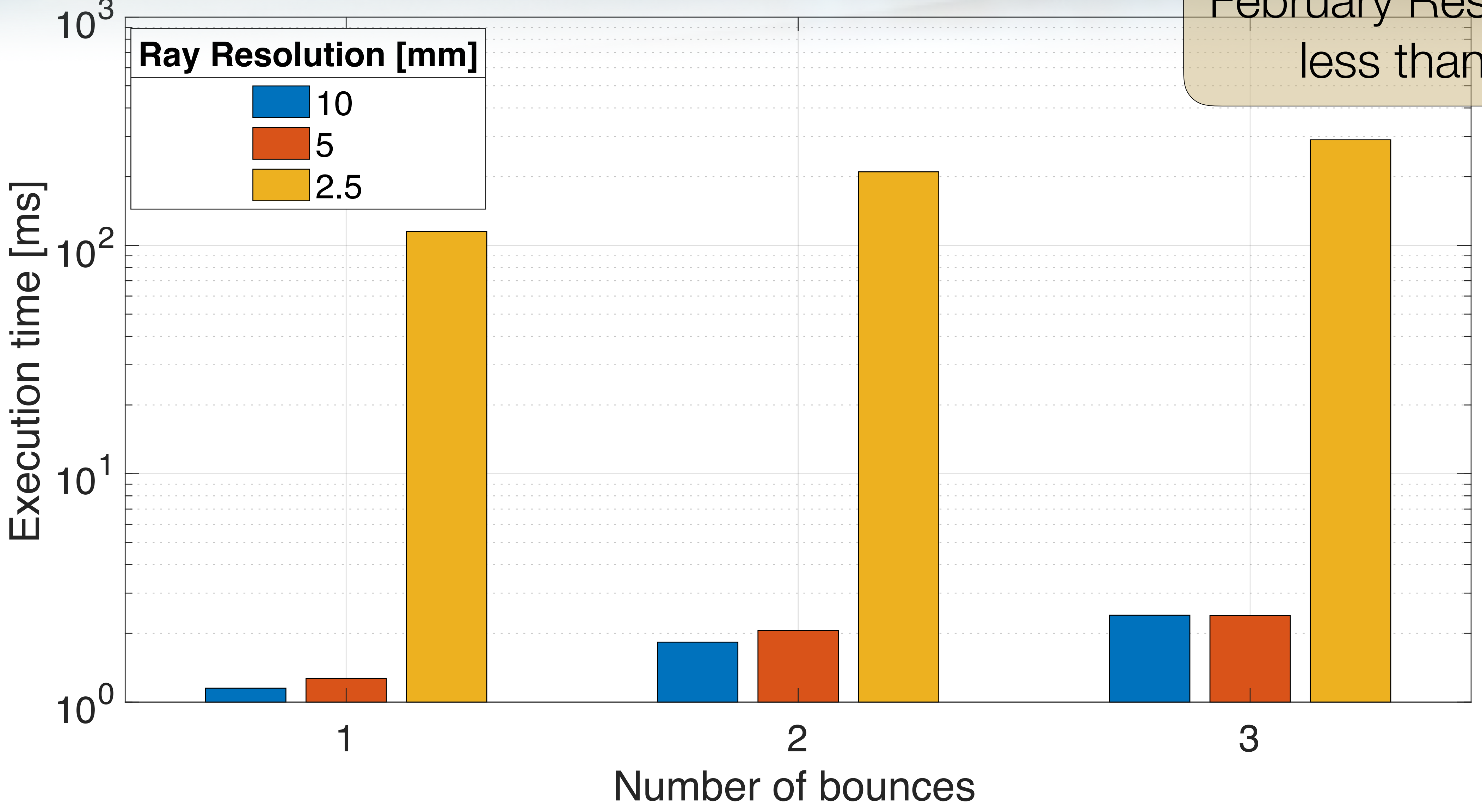
Ray Bounce Illustration

P. Kenneally, “Faster than Real-Time GPGPU Radiation Pressure Modeling Methods,” *Doctoral Dissertation, University of Colorado, Boulder, May 2019.*



(a) One bounce. (b) Two bounces. (c) Image difference.

SRP Force/Torque Evaluation Comparison



February Results, now less than 1ms

Execution Times Across Different GPUs

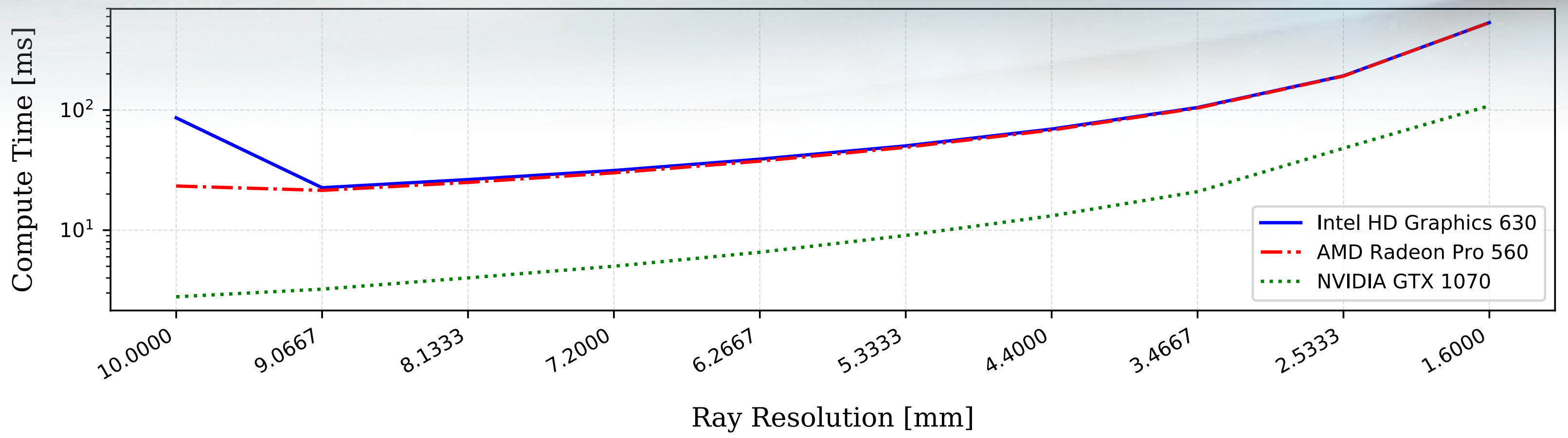


Figure 4.32: Execution times for ray resolutions from 0.01 mm to 0.0016 mm, for one bounce

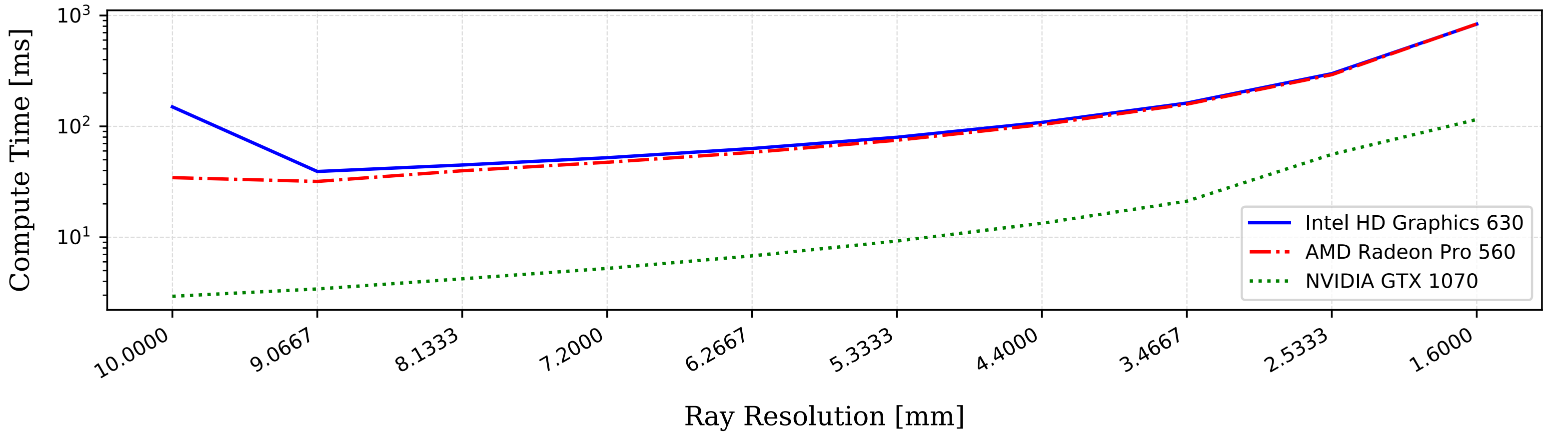


Figure 4.33: Execution times for ray resolutions from 0.01 mm to 0.0016 mm, for a maximum of three bounces.

Conclusions

- Basilisk's modular natural allows for rapid integration of space environment models
- The current implementation provides convenient base classes for atmospheric neutral density and magnetic field models, as well as the more complex MSIS and WMM models.
- The GPU-based Facet SRP modeling is going to be released shortly in the open *develop* branch. The Ray-Tracing SRP modeling is expected to be incorporated into the open Basilisk repo later in 2020.
- Information about Basilisk can be found at
  - http://hanspeterschaub.info/basilisk/

Questions?