Society of Digital Twins

Contributors:

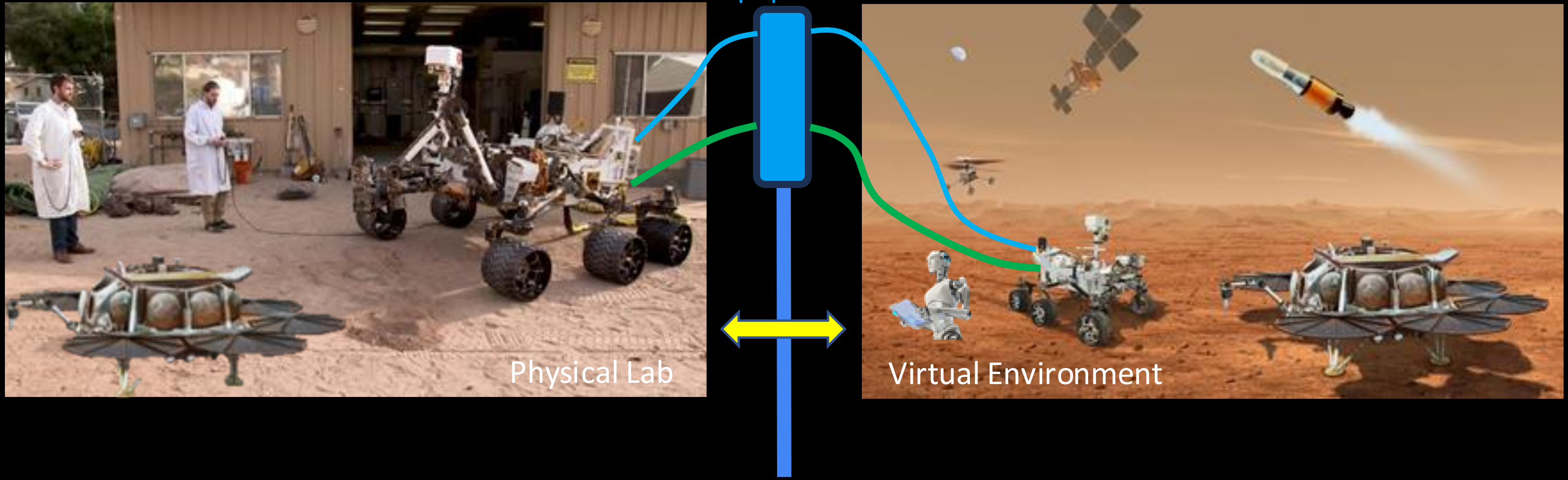
- Dr. Edward Chow, JPL
- Dr. Thomas Lu, JPL
- Dr. Bingbing Li, CSUN
- Jared Carrillo, CSUN
- Elliott Sadler, CSUN
- Neville Elieh Janvisloo, CSUN
- Subhobrata Chakraborty, CSUN
- Rayyan Mridha, Northeastern University
- Luke Cortez, USC
- Leon Gold, University of Chicago

Web 3.0 Metaverse Virtual Test Lab

(1) Real World Test Environment

Test Equipment

(2) True to Life Virtual Test Environment for Distributed Partners

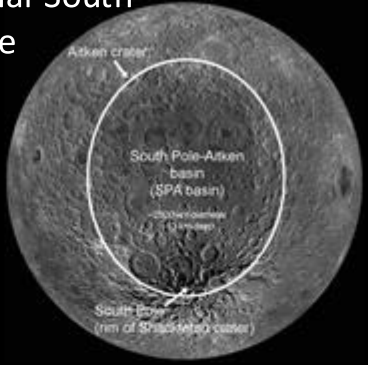


Physical Testing Planned through Virtual Environment: Cost reduction and testing the impossible

Virtual Test Assisted/Validated with Physical Test Data: Better model through differentiable learning

Virtual Environment for Collaborative Lunar Explorations (VIRCLE)

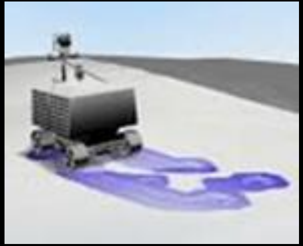
Lunar South Pole

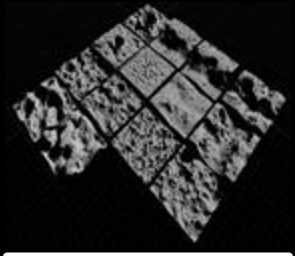

Aitken crater
South Pole-Aitken basin (SPA basin)
South Pole (rim of Shackleton crater)

High Fidelity Virtual Test Environment

Physical Digital Twins Testbeds

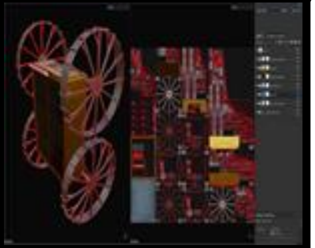
Physics Simulation Tools





Terrain Library

Vehicle CAD models



AR/VR Visualization Tools

Shared Tool Library

Shared Model Database

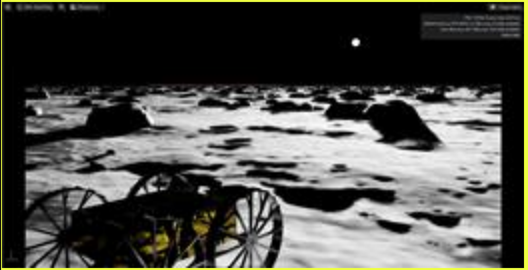
3D Streaming Cloud Infrastructure

VIRCLE

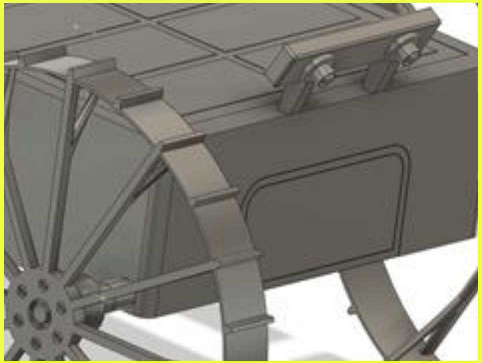
Omniverse

Rovers built in:

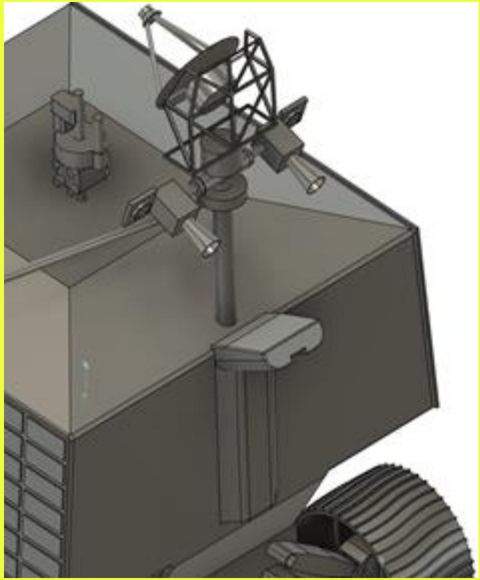
- Fusion 360 for general model
- 3DS Max for topology and meshing
- Adobe Substance for texturing
- Omniverse for physics and movement

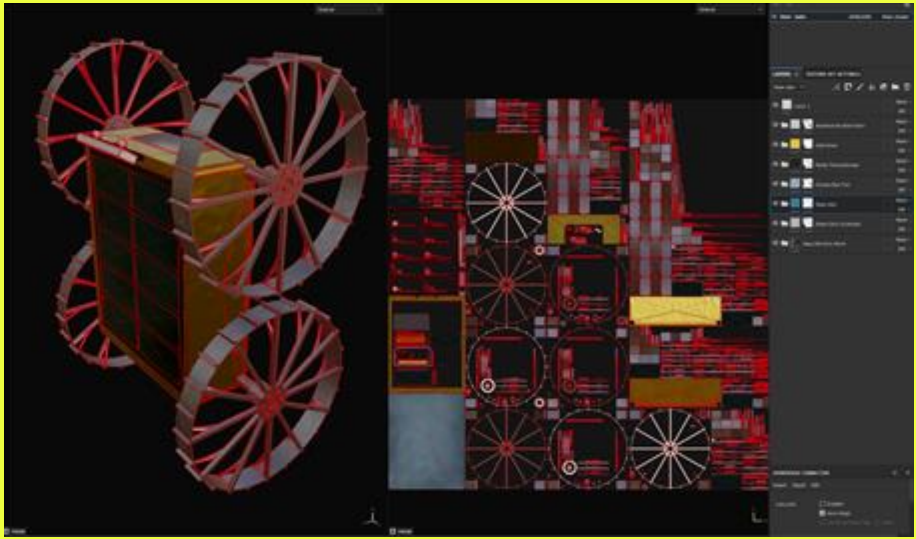


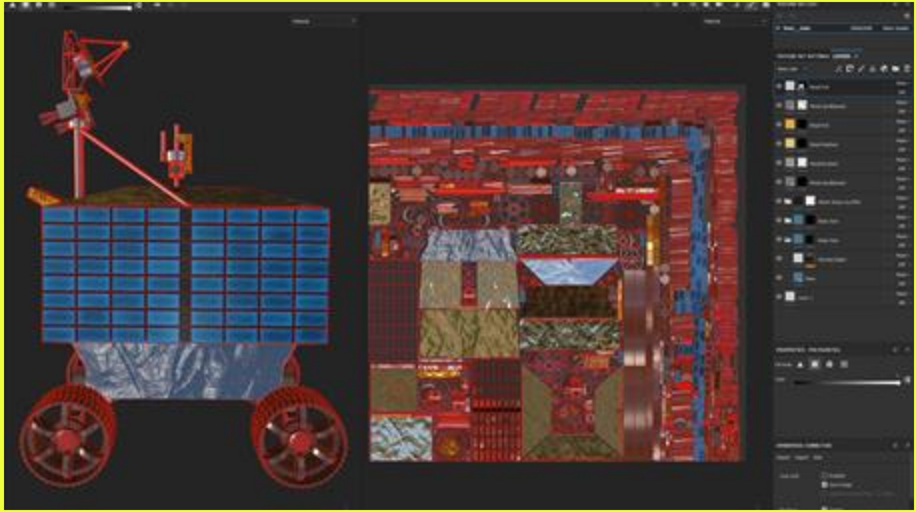
CADRE



VIPER







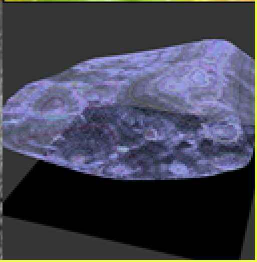
Omniverse

Large scale terrain is based on NASA LRO & LOLA.

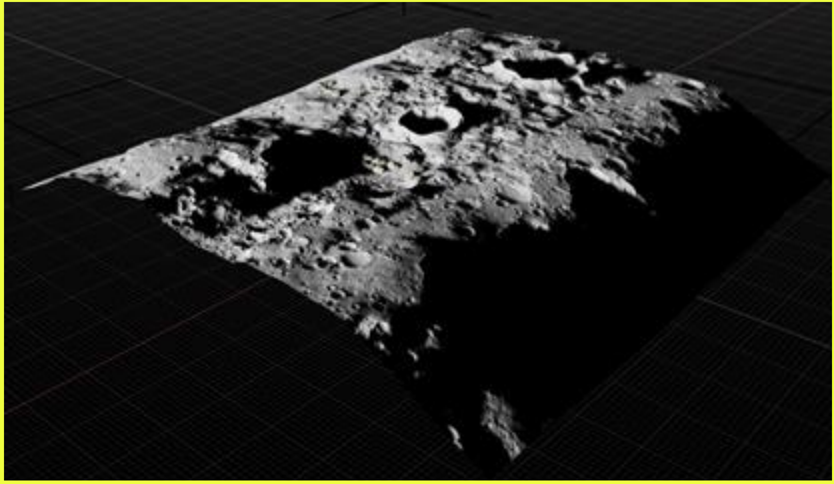
High-Definition lunar terrain created based on research and artists renderings.

Models created in 3DS Max. Terrain and rocks were created to be procedurally generated

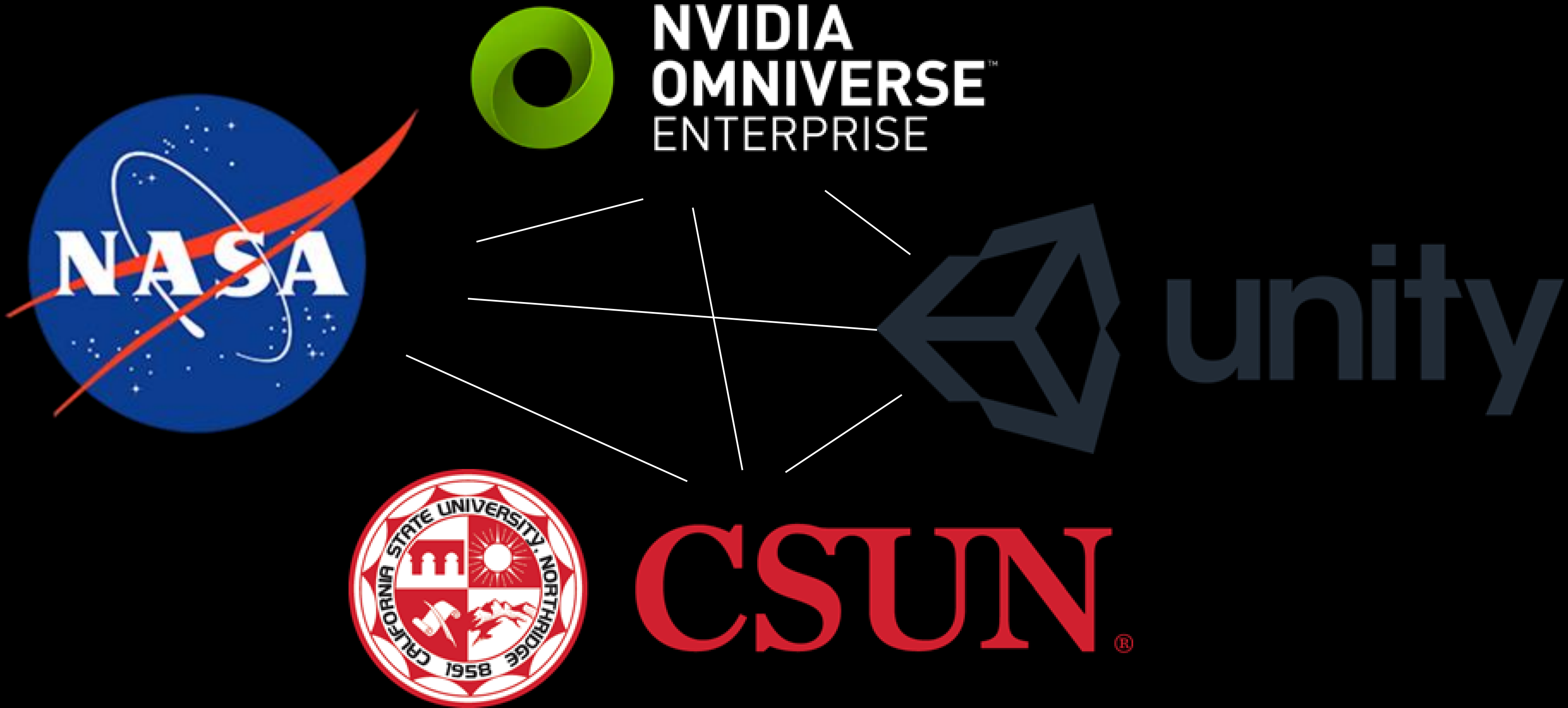
Lunar Orbiter Data



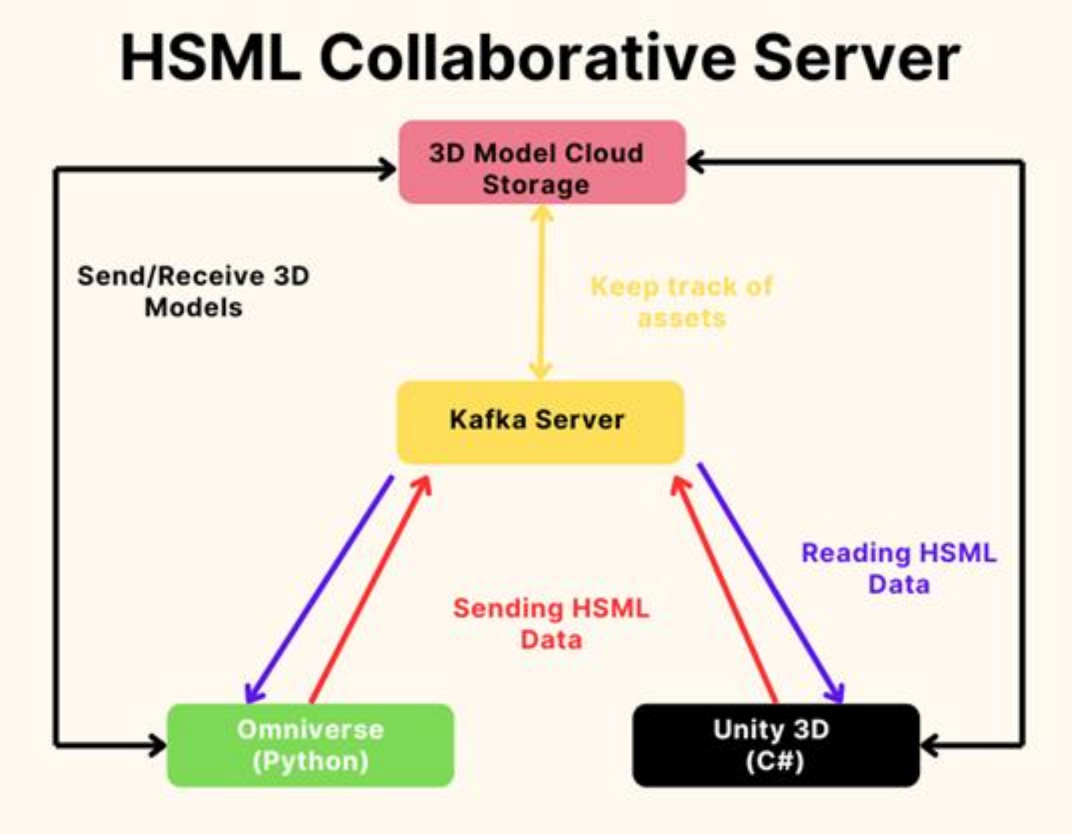
Realistically Modeled Terrain



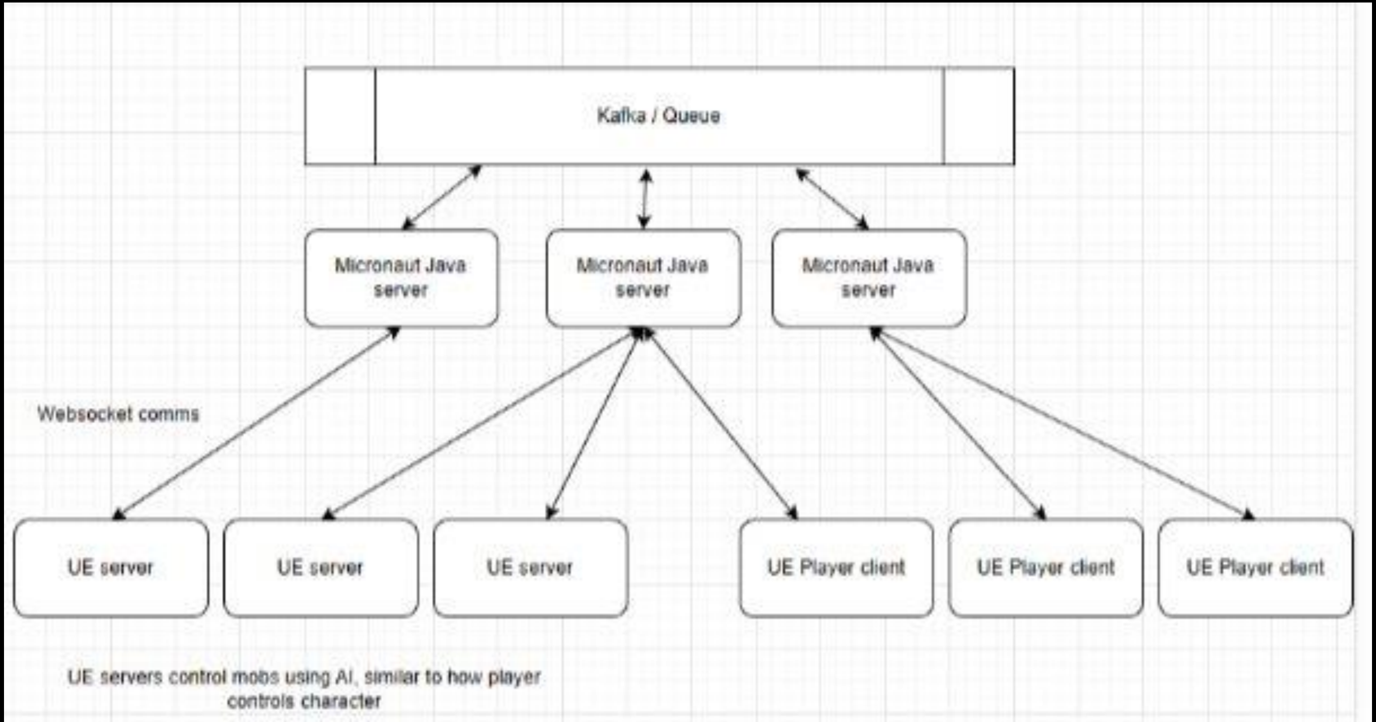
Collaboration Between Organizations


NVIDIA OMNIVERSE™ ENTERPRISE
NASA
unity
CALIFORNIA STATE UNIVERSITY, NORTHRIDGE 1958
CSUN®

Current Architecture Diagram



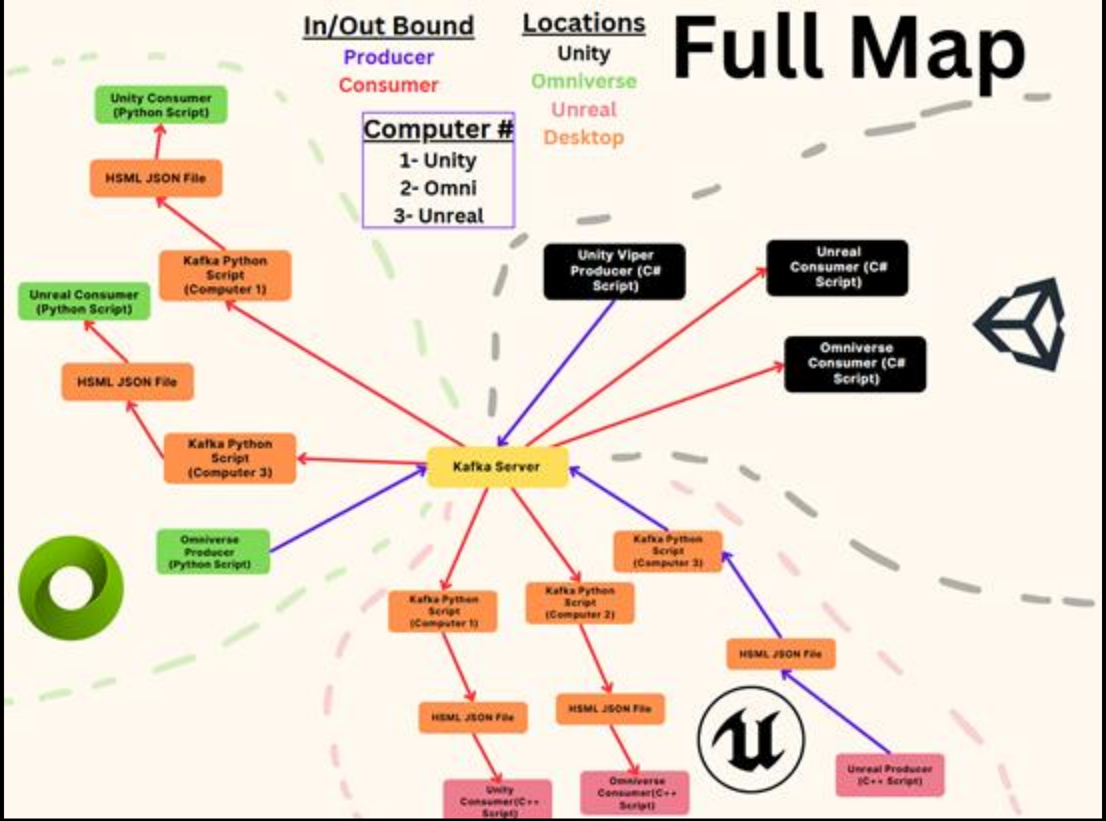
Future Expanded Architecture Diagram



Websocket comms

UE servers control mobs using AI, similar to how player controls character

Full Architecture Diagram



Full Map

In/Out Bound
Producer
Consumer

Locations
Unity
Omniverse
Unreal
Desktop

Computer #
1- Unity
2- Omni
3- Unreal

Unity Consumer (Python Script)

HSML JSON File

Kafka Python Script (Computer 1)

Unreal Consumer (Python Script)

HSML JSON File

Kafka Python Script (Computer 3)

Omniverse Producer (Python Script)

Kafka Server

Unity Viper Producer (C# Script)

Unreal Consumer (C# Script)

Omniverse Consumer (C# Script)

Kafka Python Script (Computer 1)

HSML JSON File

Unity Consumer(C++ Script)

Kafka Python Script (Computer 2)

HSML JSON File

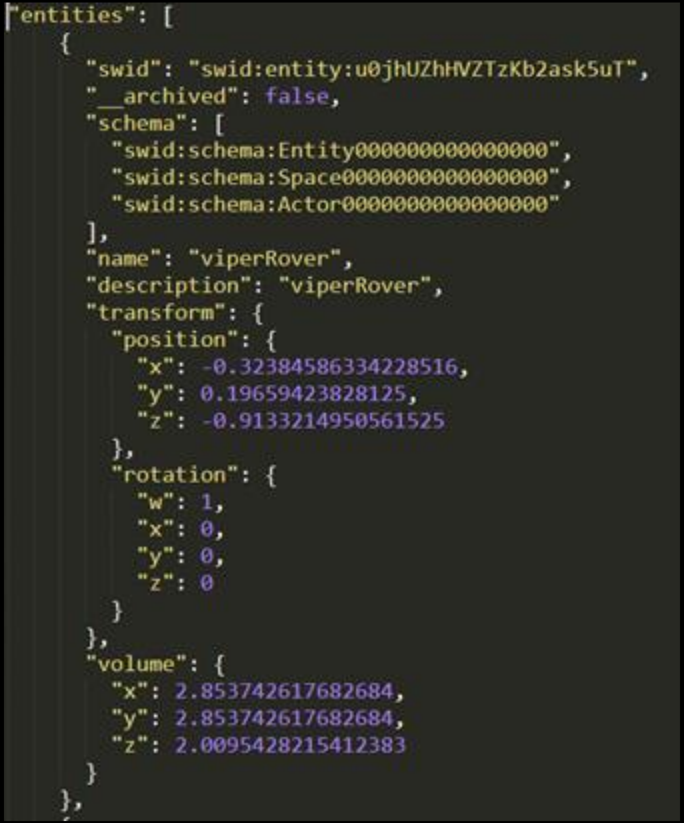
Omniverse Consumer(C++ Script)

Kafka Python Script (Computer 3)

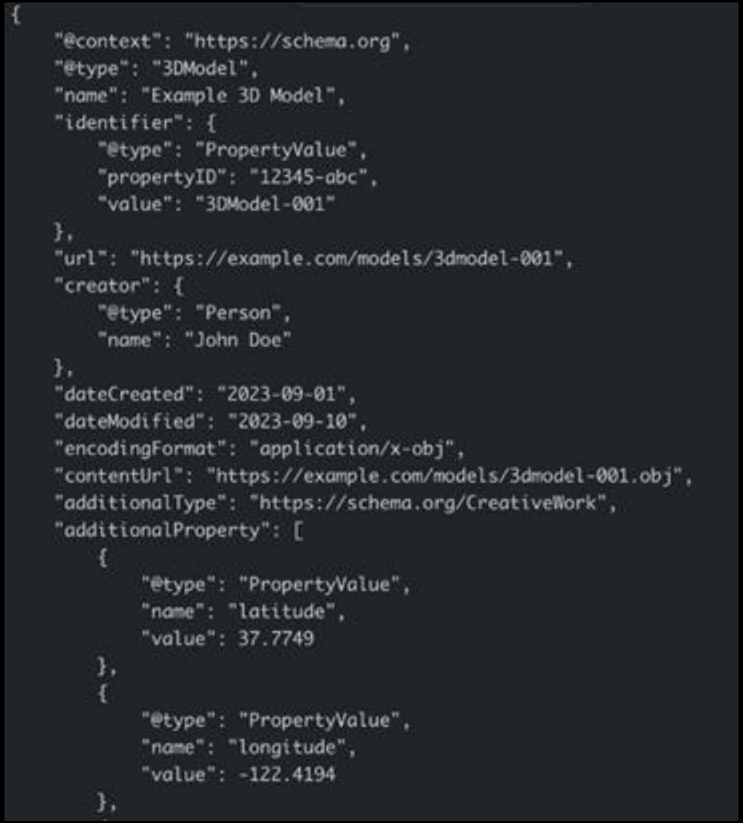
HSML JSON File

Unreal Producer (C++ Script)

HSML Schema Example (Data Transfered)

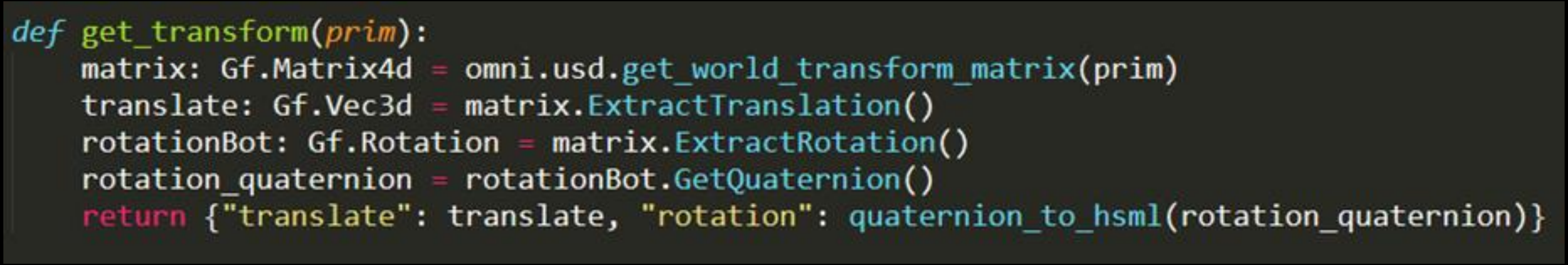


```
"entities": [
    {
      "swid": "swid:entity:u0jhUZhHVZTzKb2ask5uT",
      "__archived": false,
      "schema": [
        "swid:schema:Entity000000000000000",
        "swid:schema:Space0000000000000000",
        "swid:schema:Actor0000000000000000"
      ],
      "name": "viperRover",
      "description": "viperRover",
      "transform": {
        "position": {
          "x": -0.32384586334228516,
          "y": 0.19659423828125,
          "z": -0.9133214950561525
        },
        "rotation": {
          "w": 1,
          "x": 0,
          "y": 0,
          "z": 0
        }
      },
      "volume": {
        "x": 2.853742617682684,
        "y": 2.853742617682684,
        "z": 2.0095428215412383
      }
    },
```



```
{
    "@context": "https://schema.org",
    "@type": "3DModel",
    "name": "Example 3D Model",
    "identifier": {
        "@type": "PropertyValue",
        "propertyID": "12345-abc",
        "value": "3DModel-001"
    },
    "url": "https://example.com/models/3dmodel-001",
    "creator": {
        "@type": "Person",
        "name": "John Doe"
    },
    "dateCreated": "2023-09-01",
    "dateModified": "2023-09-10",
    "encodingFormat": "application/x-obj",
    "contentUrl": "https://example.com/models/3dmodel-001.obj",
    "additionalType": "https://schema.org/CreativeWork",
    "additionalProperty": [
        {
            "@type": "PropertyValue",
            "name": "latitude",
            "value": 37.7749
        },
        {
            "@type": "PropertyValue",
            "name": "longitude",
            "value": -122.4194
        },
```

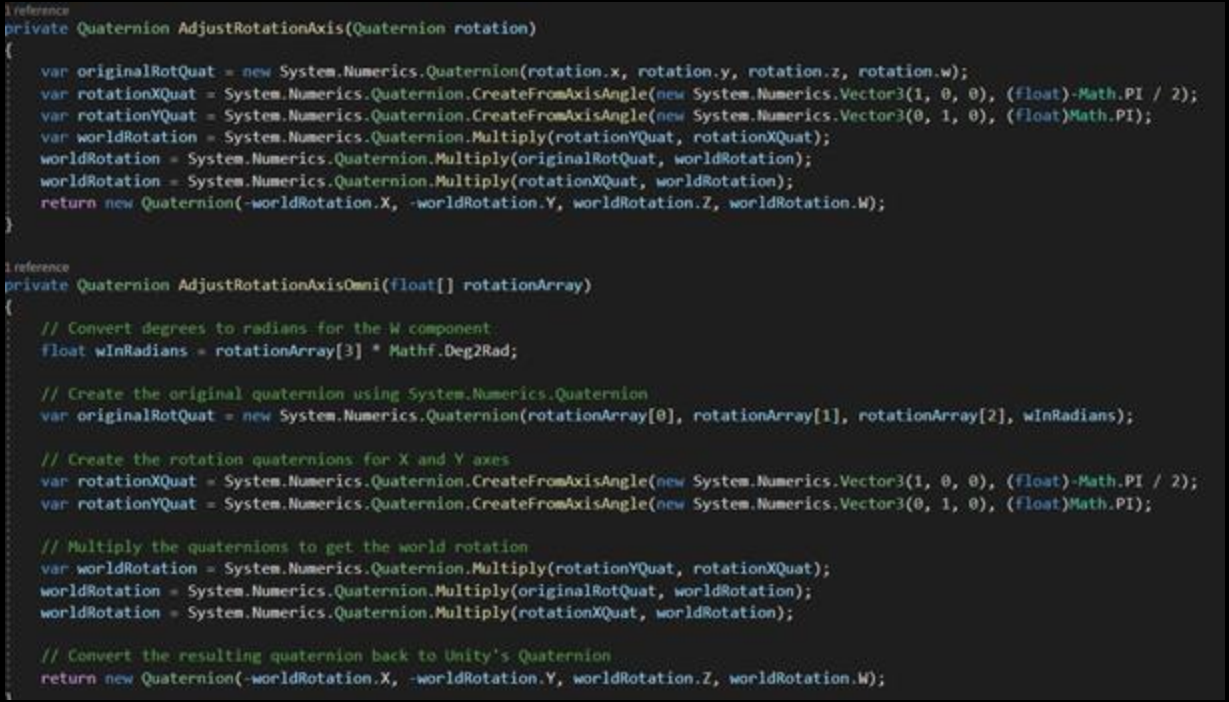
Extract Data from Digital Twins (xyz, rotations, etc)



```python
def get_transform(prim):
    matrix: Gf.Matrix4d = omni.usd.get_world_transform_matrix(prim)
    translate: Gf.Vec3d = matrix.ExtractTranslation()
    rotationBot: Gf.Rotation = matrix.ExtractRotation()
    rotation_quaternion = rotationBot.GetQuaternion()
    return {"translate": translate, "rotation": quaternion_to_hsml(rotation_quaternion)}
```

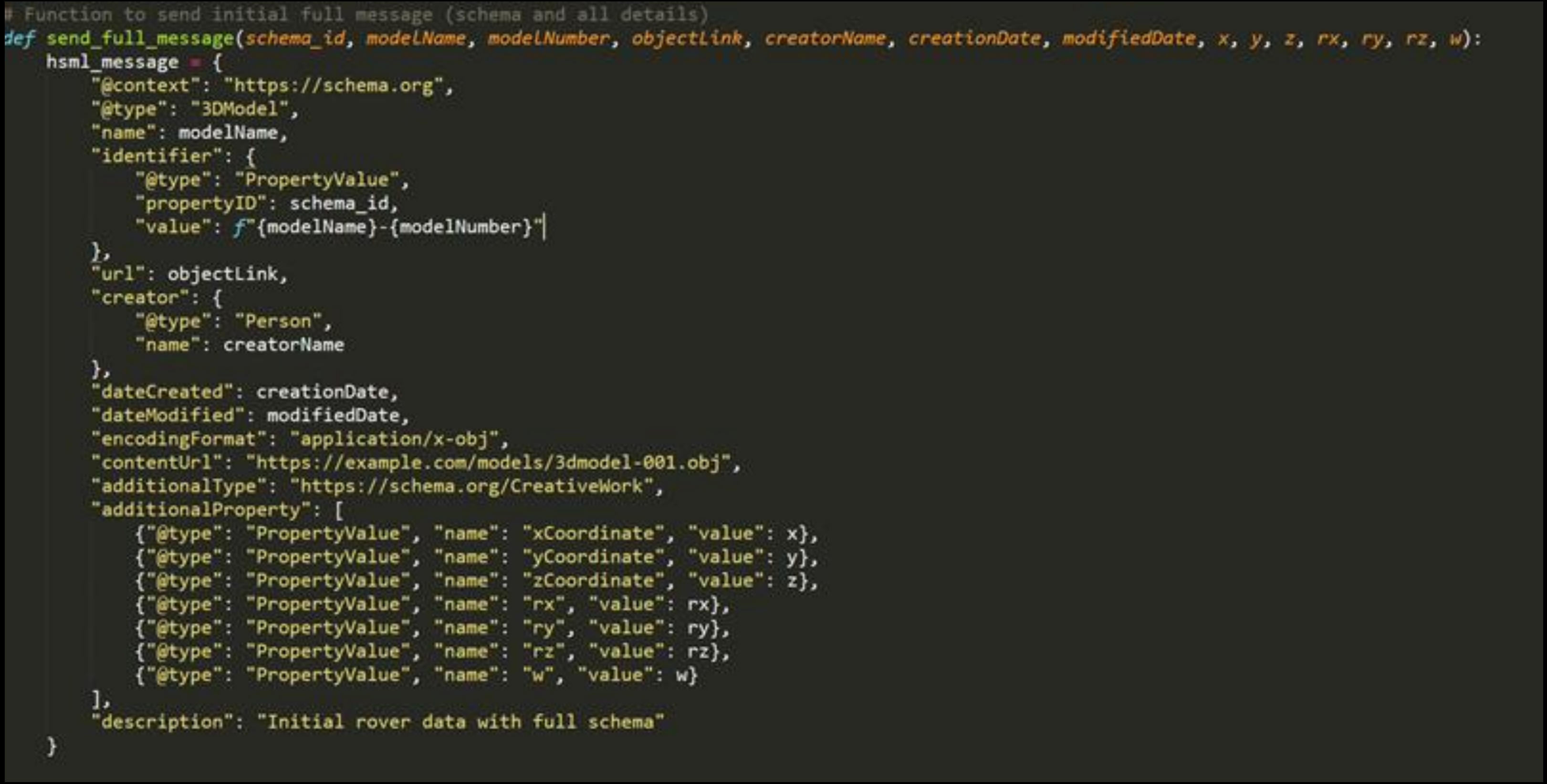
Coordinate System Matrix Converter

- Programs have different coordinate systems (right hand/left hand)
- Quaternions affected by flipped Y & Z values



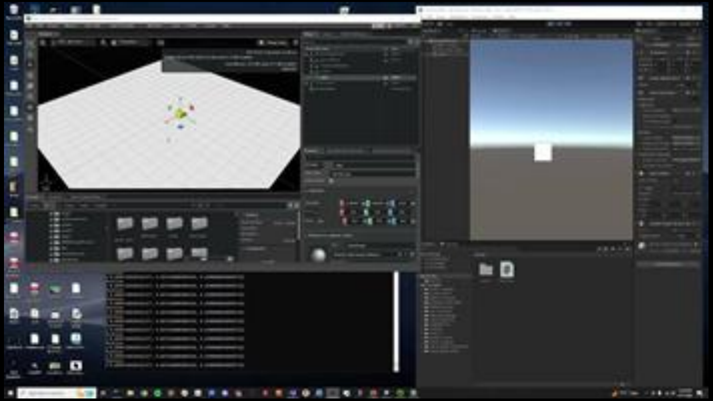
```
1 reference
private Quaternion AdjustRotationAxis(Quaternion rotation)
{
    var originalRotQuat = new System.Numerics.Quaternion(rotation.x, rotation.y, rotation.z, rotation.w);
    var rotationXQuat = System.Numerics.Quaternion.CreateFromAxisAngle(new System.Numerics.Vector3(1, 0, 0), (float)-Math.PI / 2);
    var rotationYQuat = System.Numerics.Quaternion.CreateFromAxisAngle(new System.Numerics.Vector3(0, 1, 0), (float)Math.PI);
    var worldRotation = System.Numerics.Quaternion.Multiply(rotationYQuat, rotationXQuat);
    worldRotation = System.Numerics.Quaternion.Multiply(originalRotQuat, worldRotation);
    worldRotation = System.Numerics.Quaternion.Multiply(rotationXQuat, worldRotation);
    return new Quaternion(-worldRotation.X, -worldRotation.Y, worldRotation.Z, worldRotation.W);
}

1 reference
private Quaternion AdjustRotationAxisOmni(float[] rotationArray)
{
    // Convert degrees to radians for the W component
    float wInRadians = rotationArray[3] * Mathf.Deg2Rad;

    // Create the original quaternion using System.Numerics.Quaternion
    var originalRotQuat = new System.Numerics.Quaternion(rotationArray[0], rotationArray[1], rotationArray[2], wInRadians);

    // Create the rotation quaternions for X and Y axes
    var rotationXQuat = System.Numerics.Quaternion.CreateFromAxisAngle(new System.Numerics.Vector3(1, 0, 0), (float)-Math.PI / 2);
    var rotationYQuat = System.Numerics.Quaternion.CreateFromAxisAngle(new System.Numerics.Vector3(0, 1, 0), (float)Math.PI);

    // Multiply the quaternions to get the world rotation
    var worldRotation = System.Numerics.Quaternion.Multiply(rotationYQuat, rotationXQuat);
    worldRotation = System.Numerics.Quaternion.Multiply(originalRotQuat, worldRotation);
    worldRotation = System.Numerics.Quaternion.Multiply(rotationXQuat, worldRotation);

    // Convert the resulting quaternion back to Unity's Quaternion
    return new Quaternion(-worldRotation.X, -worldRotation.Y, worldRotation.Z, worldRotation.W);
```

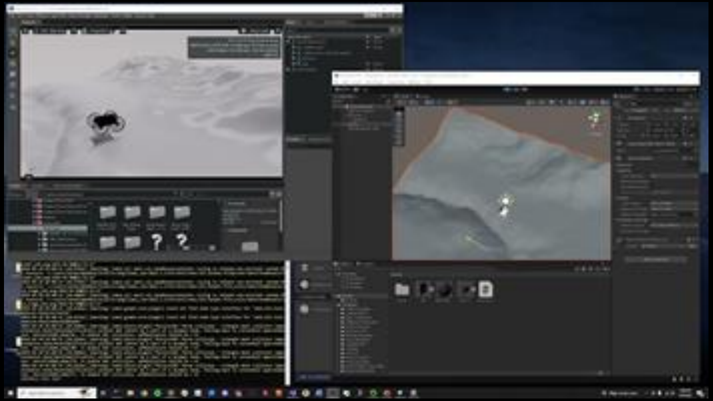
HSML Standard Formatting

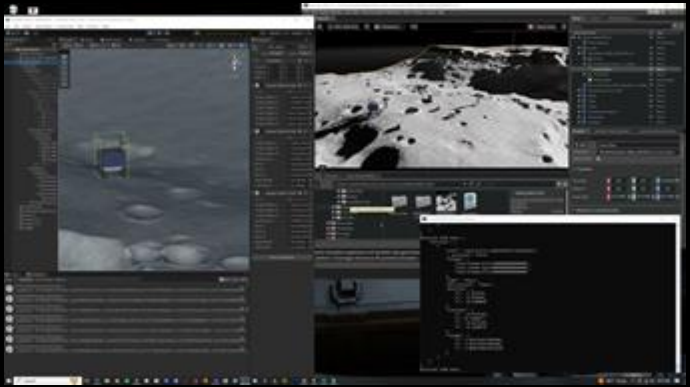


```
# Function to send initial full message (schema and all details)
def send_full_message(schema_id, modelName, modelNumber, objectLink, creatorName, creationDate, modifiedDate, x, y, z, rx, ry, rz, w):
    hsml_message = {
        "@context": "https://schema.org",
        "@type": "3DModel",
        "name": modelName,
        "identifier": {
            "@type": "PropertyValue",
            "propertyID": schema_id,
            "value": f"{modelName}-{modelNumber}"
        },
        "url": objectLink,
        "creator": {
            "@type": "Person",
            "name": creatorName
        },
        "dateCreated": creationDate,
        "dateModified": modifiedDate,
        "encodingFormat": "application/x-obj",
        "contentUrl": "https://example.com/models/3dmodel-001.obj",
        "additionalType": "https://schema.org/CreativeWork",
        "additionalProperty": [
            {"@type": "PropertyValue", "name": "xCoordinate", "value": x},
            {"@type": "PropertyValue", "name": "yCoordinate", "value": y},
            {"@type": "PropertyValue", "name": "zCoordinate", "value": z},
            {"@type": "PropertyValue", "name": "rx", "value": rx},
            {"@type": "PropertyValue", "name": "ry", "value": ry},
            {"@type": "PropertyValue", "name": "rz", "value": rz},
            {"@type": "PropertyValue", "name": "w", "value": w}
        ],
        "description": "Initial rover data with full schema"
    }
```

TCP/JSON Server POC

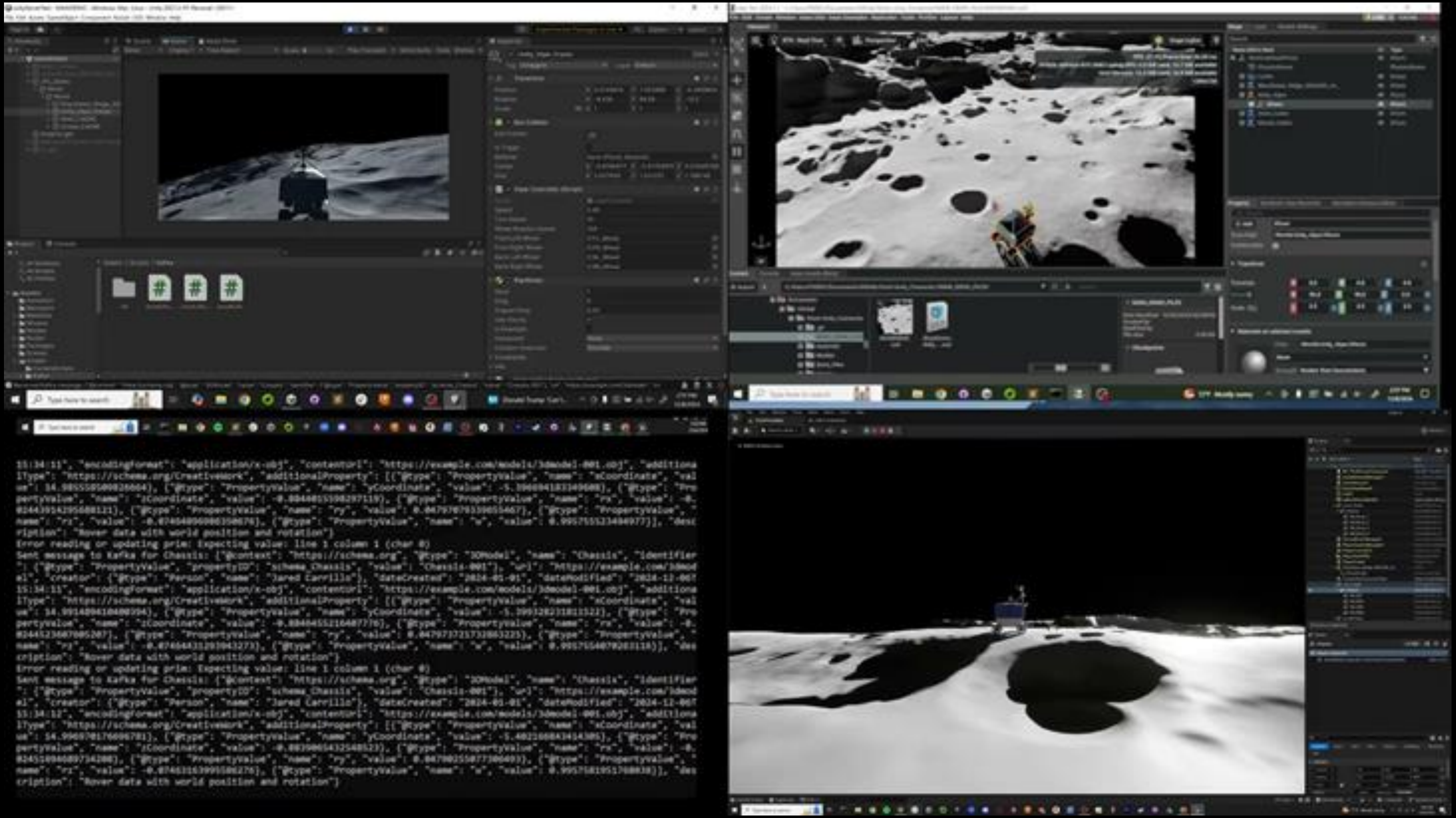






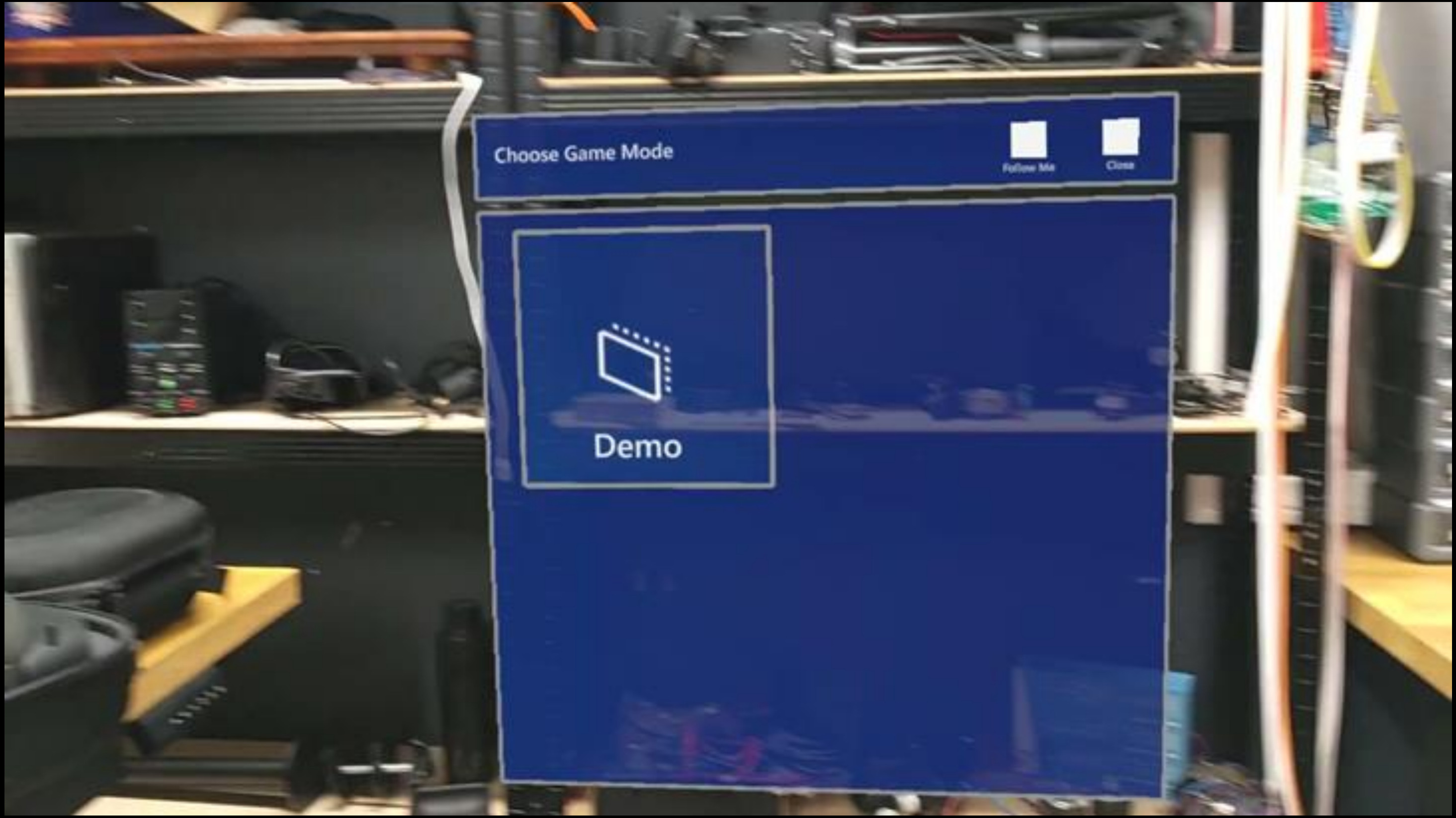
KAFKA HSML Server Progress

Final HSML w/ all 3 Programs



```
15:34:11", "encodingFormat": "application/x-obj", "contentUrl": "https://example.com/models/3dmodel-001.obj", "additiona
lType": "https://schema.org/CreativeWork", "additionalProperty": [{"@type": "PropertyValue", "name": "xCoordinate", "val
ue": 14.985555000026604}, {"@type": "PropertyValue", "name": "yCoordinate", "value": -5.396694183349608}, {"@type": "Pro
pertyValue", "name": "zCoordinate", "value": -0.0844815598297119}, {"@type": "PropertyValue", "name": "rx", "value": -0.
024439142956080121}, {"@type": "PropertyValue", "name": "ry", "value": 0.04797070339055467}, {"@type": "PropertyValue", "
name": "rz", "value": -0.07404809808350676}, {"@type": "PropertyValue", "name": "w", "value": 0.995755523404977}], "desc
ription": "Rover data with world position and rotation"}
Error reading or updating prim: Expecting value: line 1 column 1 (char 0)
Sent message to Kafka for Chassis: {"@context": "https://schema.org", "@type": "3DModel", "name": "Chassis", "identifier
": {"@type": "PropertyValue", "propertyID": "schema_Chassis", "value": "Chassis-001"}, "url": "https://example.com/3dmod
el", "creator": {"@type": "Person", "name": "Jared Carrillo"}, "dateCreated": "2024-01-01", "dateModified": "2024-12-06T
15:34:11", "encodingFormat": "application/x-obj", "contentUrl": "https://example.com/models/3dmodel-001.obj", "additiona
lType": "https://schema.org/CreativeWork", "additionalProperty": [{"@type": "PropertyValue", "name": "xCoordinate", "val
ue": 14.991480410400394}, {"@type": "PropertyValue", "name": "yCoordinate", "value": -5.399328231811522}, {"@type": "Pro
pertyValue", "name": "zCoordinate", "value": -0.0844455216407776}, {"@type": "PropertyValue", "name": "rx", "value": -0.
024452368070085207}, {"@type": "PropertyValue", "name": "ry", "value": 0.04797372573286322}, {"@type": "PropertyValue", "
name": "rz", "value": -0.07404431293043273}, {"@type": "PropertyValue", "name": "w", "value": 0.9957554070203118}], "des
cription": "Rover data with world position and rotation"}
Error reading or updating prim: Expecting value: line 1 column 1 (char 0)
Sent message to Kafka for Chassis: {"@context": "https://schema.org", "@type": "3DModel", "name": "Chassis", "identifier
": {"@type": "PropertyValue", "propertyID": "schema_Chassis", "value": "Chassis-001"}, "url": "https://example.com/3dmod
el", "creator": {"@type": "Person", "name": "Jared Carrillo"}, "dateCreated": "2024-01-01", "dateModified": "2024-12-06T
15:34:12", "encodingFormat": "application/x-obj", "contentUrl": "https://example.com/models/3dmodel-001.obj", "additiona
lType": "https://schema.org/CreativeWork", "additionalProperty": [{"@type": "PropertyValue", "name": "xCoordinate", "val
ue": 14.996970176006781}, {"@type": "PropertyValue", "name": "yCoordinate", "value": -5.402166843414305}, {"@type": "Pro
pertyValue", "name": "zCoordinate", "value": -0.0839065432548523}, {"@type": "PropertyValue", "name": "rx", "value": -0.
024510940089734208}, {"@type": "PropertyValue", "name": "ry", "value": 0.04790255077300409}, {"@type": "PropertyValue", "
name": "rz", "value": -0.07463163995586276}, {"@type": "PropertyValue", "name": "w", "value": 0.9957581951768038}], "des
cription": "Rover data with world position and rotation"}
```

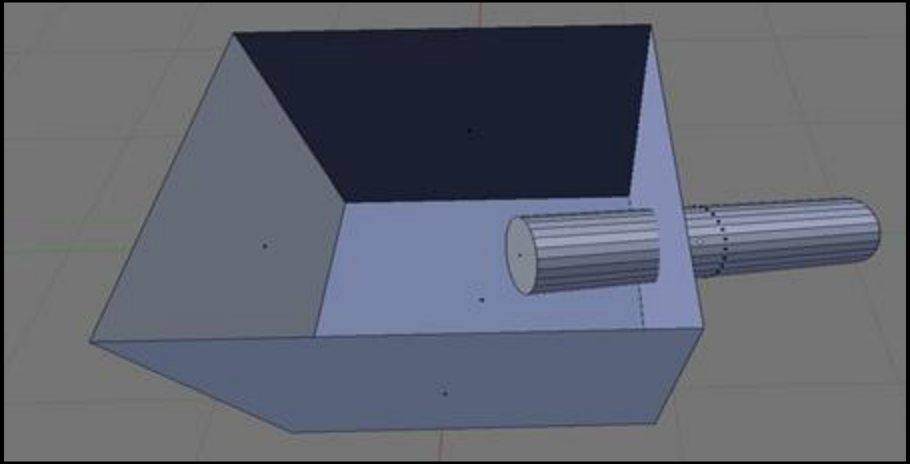
Moonwalker AR Collaboration Program



What are 3D models?

A 3D model is a **digital** representation of an **object** or scene that is created using specialized computer software

- 3D Models are visual only (think of a ghost)
- They do not interact with anything
- Mathematically they are only points in spaces (a mesh)

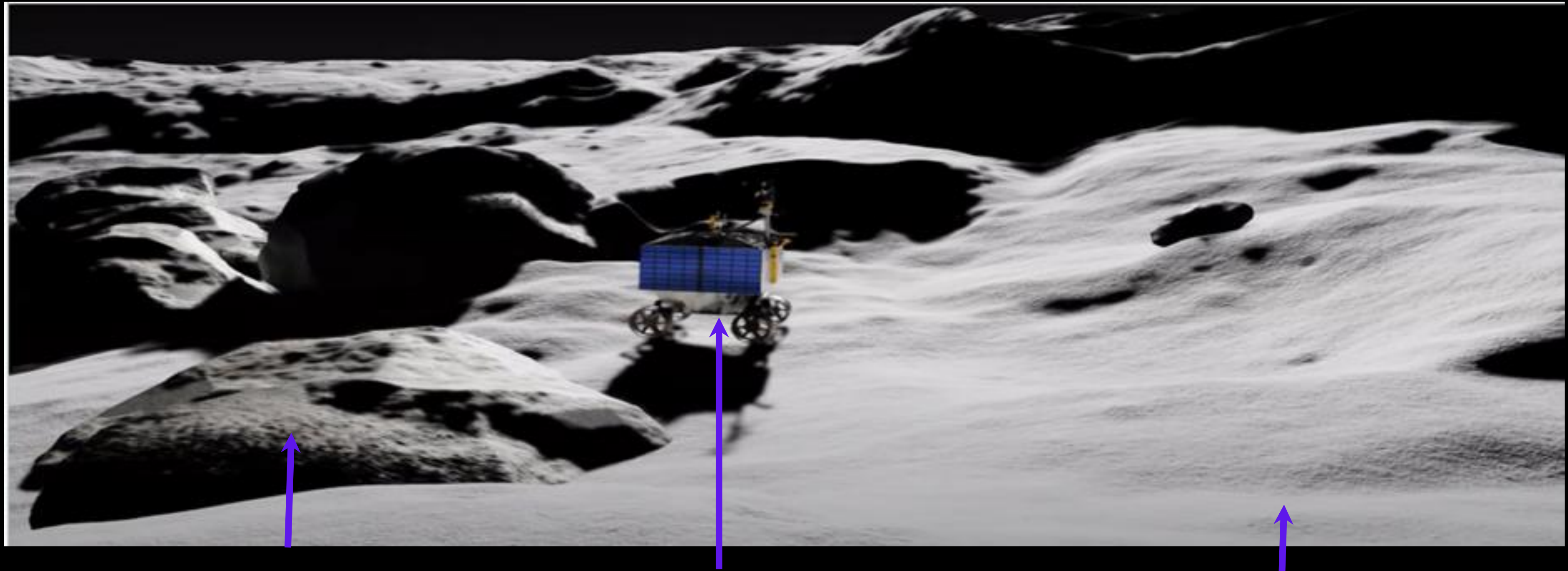


How do we make 3D models respond to physics?

Collider: an invisible component attached to a game object that defines its shape for the purpose of detecting collisions with other objects

- Adding a collider to the 3D model tells program our model has importance and lets us control how it responds to things like gravity or interactions

Current Digital World (Game Map)



Rock 3D Model

- Collider

Viper 3D Model

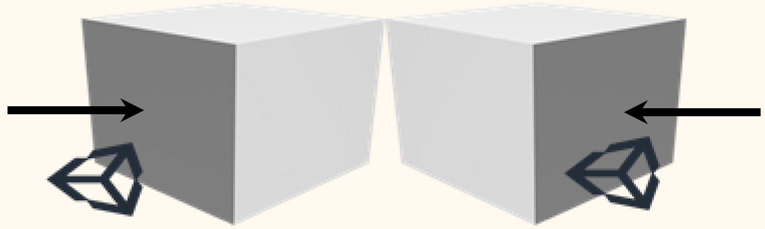
- Collider (Wheels/chassis)
- Gravity

Terrain 3D Model

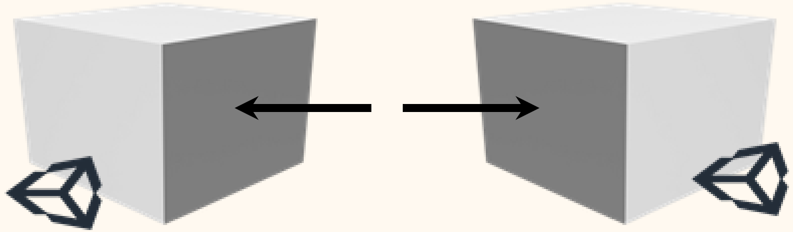
- Collider

What does that mean when it comes to simulating “physical” crashes in our system?

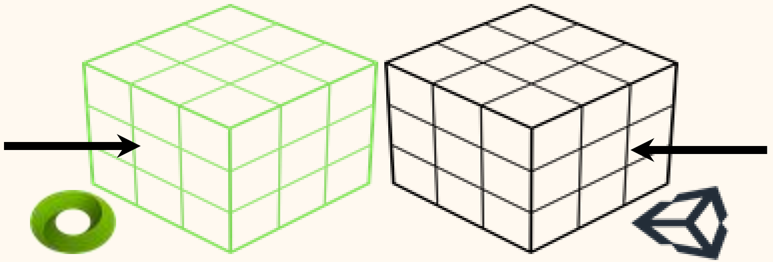
Action: Crash



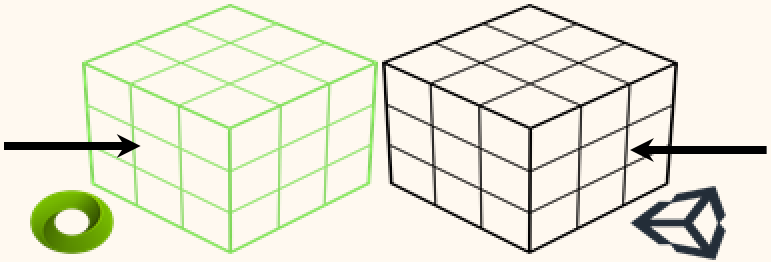
Reaction: Bounce Off



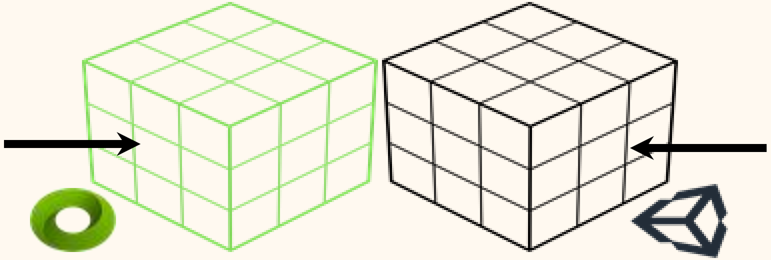
Action: Crash



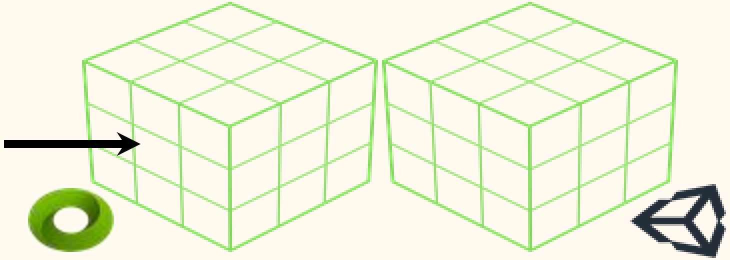
Who takes priority ???



Potential Work Around



Upon contact biggest force/mass is calculated



Priority is transferred to winner and parent/child relationship is reversed

- Basic idea is that by turning off and on colliders in each program this can create a way to change parent/child relationships

Final HSML w/ Collisions

Conclusions

- Successfully created a cross-platform synchronization system between Unity and Omniverse Isaac Sim using Kafka and structured HSML-style JSON.
- Developed a clean, modular producer-consumer pattern using an inControl flag to manage state handoff.
- Demonstrated real-time control switching and accurate position/rotation mirroring between platforms.
- Implemented single-engine collision handling via Unity trigger colliders, improving stability and simplifying physics simulation.
- The system forms the foundation of a shared simulation environment—akin to a multiplayer game engine—for autonomous systems development.

Future Steps

- Kafka improvements: Optimize for lower latency and larger message throughput; introduce topic-based message routing.
- Dynamic authority switching: Add logic to toggle control based on simulation state, health, or remote user input.
- Multiple agent support: Scale beyond a single rover to support multi-agent interaction in a shared space.
- Visual diagnostics: Add UI elements or overlays to show live sync status, connection health, or control ownership.

Lessons Learned

- Cross-platform sync is hard: Different coordinate systems, unit scales, and origin points required careful alignment and iteration.
- Simplicity wins: Delegating collision handling to a single engine (Unity) simplified system complexity and avoided conflict resolution issues.
- JSON + HSML works well: Moving to structured data with semantic tags improved traceability and maintainability.
- Real-time ≠ perfect time: Accepting and designing for minor latency was key to building a robust system.
- Versioning and isolation are essential when managing two engines and a shared data bus.

Thank You