Simulating Worlds

for machine learning

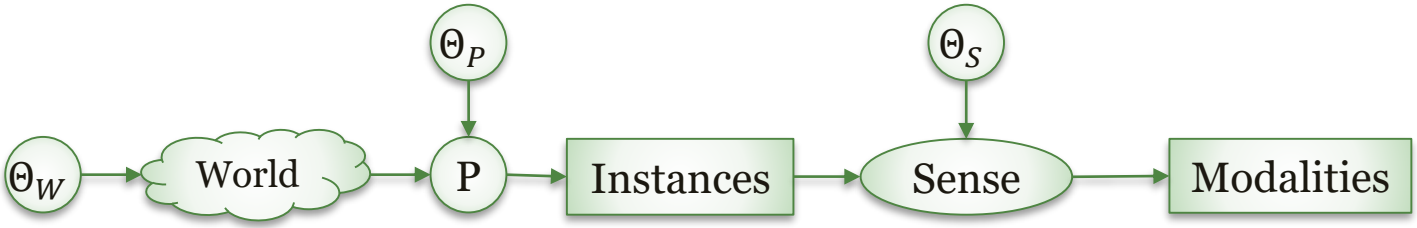
Authors: Tobias Weis, Timm Hess, and Visvanathan Ramesh

Content

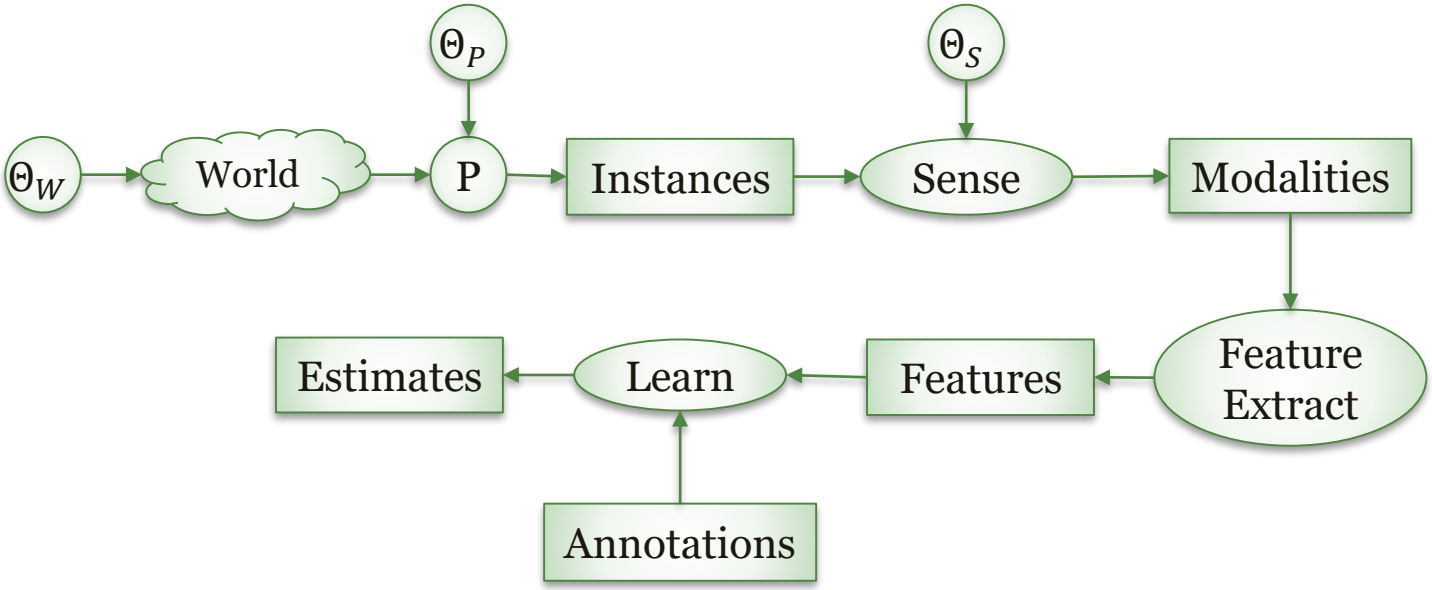
- Why, when, and how to simulate data
- Simulation Software Workflow
- Learning from Virtual Worlds

Overview ML Workflows

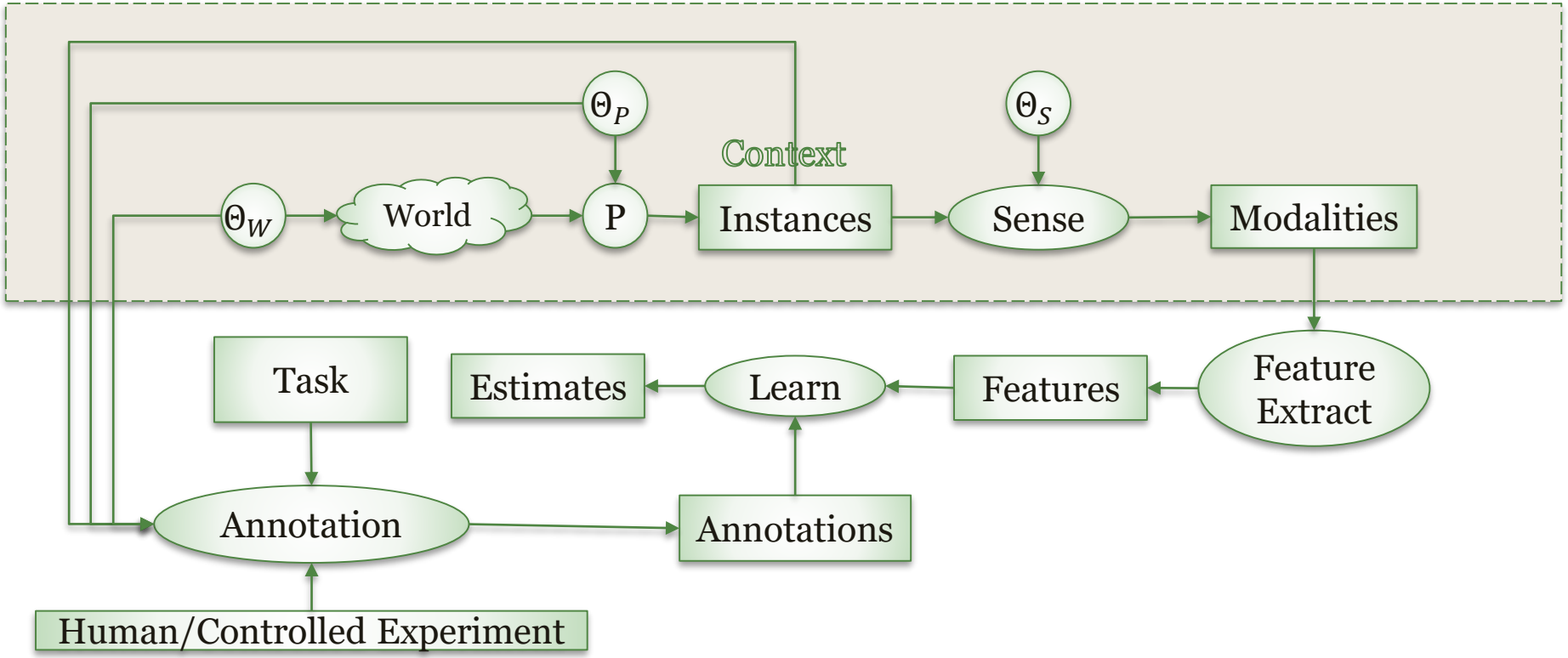
Overview ML Workflows



Overview ML Workflows



Overview ML workflows

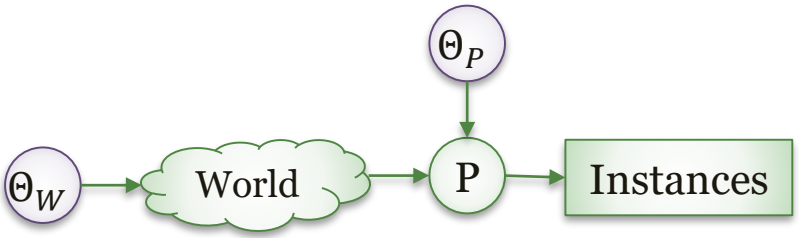


Why Simulation

- Human annotation is expensive, time-consuming, may contain errors
- Controlled experiments may be infeasible
- Sensed/captured instances may only be a subset of probable situations/scenarios

Simulation Design

The world we sense is governed by parameters:



ML wants to estimate a subset of those: **Task**
This subset may be influenced/modulated by other variables: **Context**

Simulation Design

Θ_W: ``world“-parameters:

- Geometric
  - Object positions, 3d-shapes, spatial relations
- Photometric
  - Materials and reflectances, scene lighting, atmospheric effects

Simulation Design

Θ_P: ``process“-parameters:

- An instance of the world is a specific configuration of entities, arranged according to a process P
- It‘s parameters Θ_P might include physical, social, or other laws that are imposed on entities

Simulation Design

Θ_S: ``sensing“-parameters:

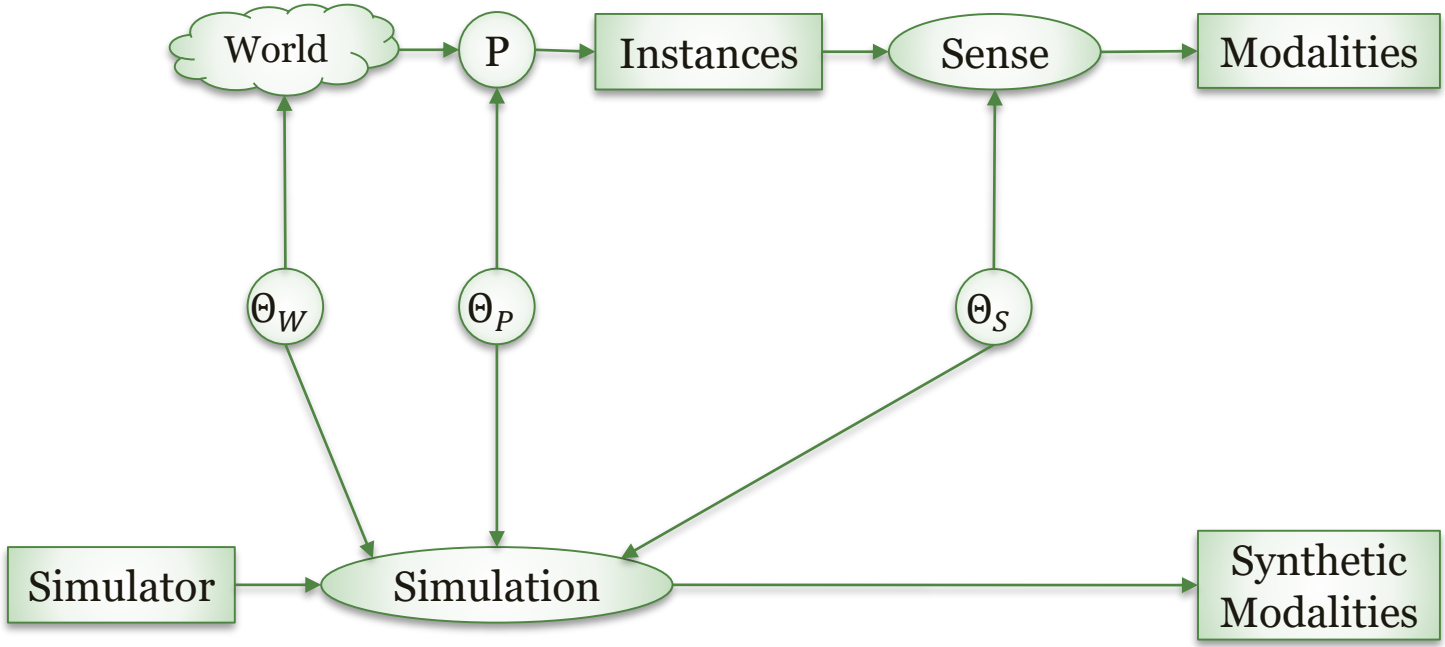
- Describe sensor-specific parameters
  - Transfer-function of physical entities to digital ones (i.e. photon->pixel-value)
  - Noise
  - Range
  - Other characteristics (i.e. lense distortions)

Simulation Design

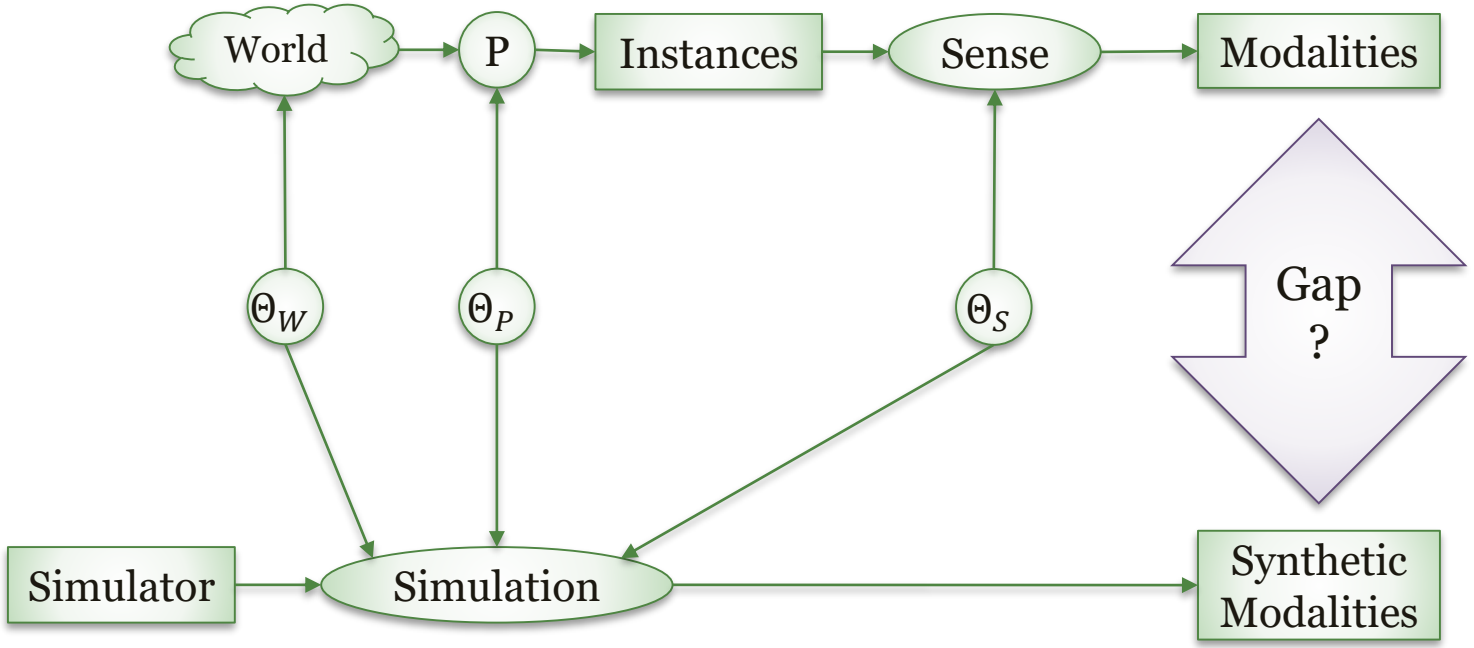
To generate useful synthetic data, we have to:

- Model parameters of interest (task)
- Model influencing parameters (context)
- Be able to synthesize needed modalities (sensor)
- Be able to generate annotations

Simulation Design



Simulation Design

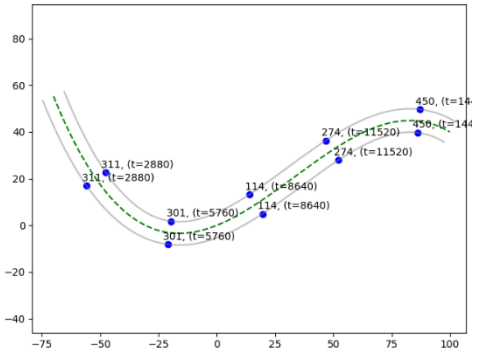


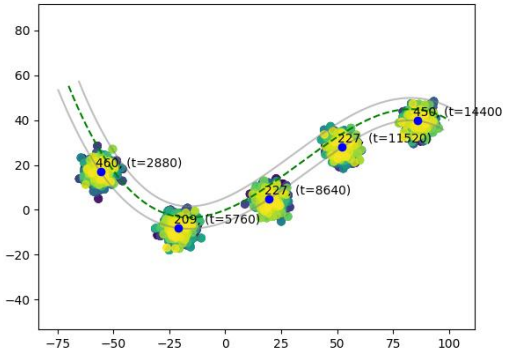
Why build Simulations?

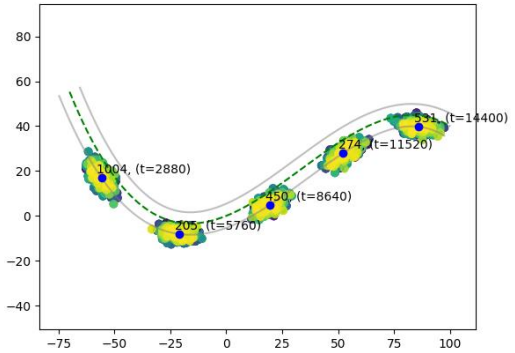
In principle, any quantity of interest can be simulated

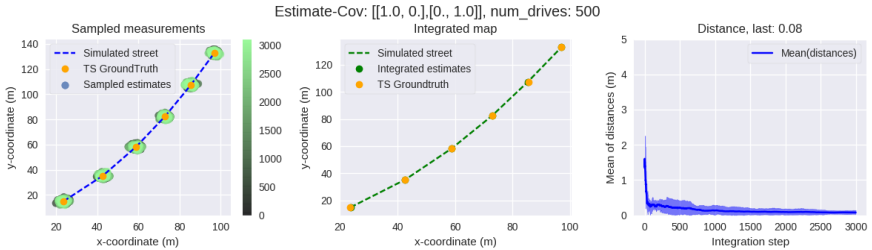
Today‘s session is focused on simulation for Computer-Vision related tasks

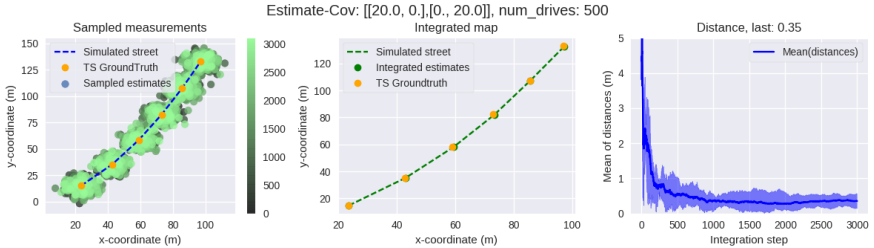
Why build Simulations?









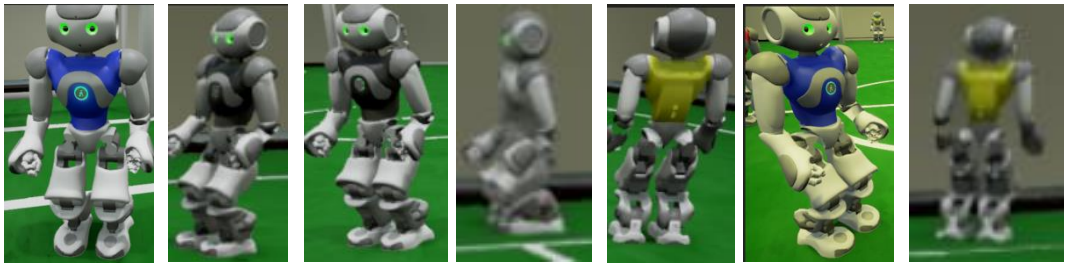


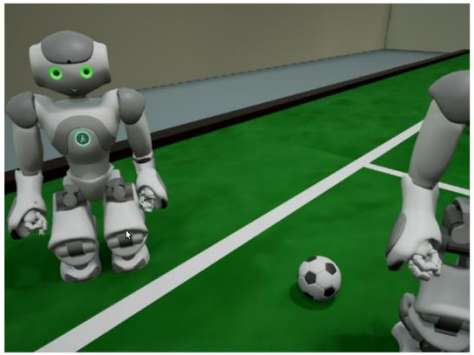
Examples
Traffic sign map integration

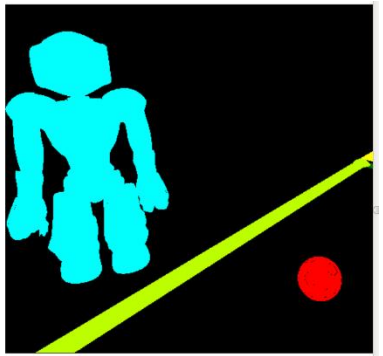
- Task
  - Integrate noisy measurements (GPS-Coordinates) into a map
- Parameters
  - Θ_W
    - Geometric: CAD-desc of streets, GPS-coordinates of TS
  - Θ_S
    - Sensor uncertainty

Why build Simulations?

- Possible source of (unlimited) situation examples – Photorealistic RGB images







Examples
RoboCup

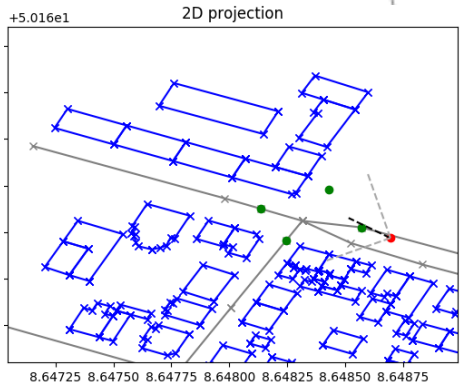
- Task
  - Object (ball, robot, line, background) classification from image patches
- Parameters
  - Θ_W
    - Geometric: CAD-desc of ball, robot, playing field
    - Photometric: textures, reflectances, external lighting (context)
  - Θ_P
    - Objects located inside field-border
    - Governed by gravity (everything on ground-plane)
    - Robots are articulated
  - Θ_S
    - Sensor resolution, possible exposure times, noise characteristics
    - Extrinsic (position,angle) and intrinsic (focal length, central point, distortion) parameters

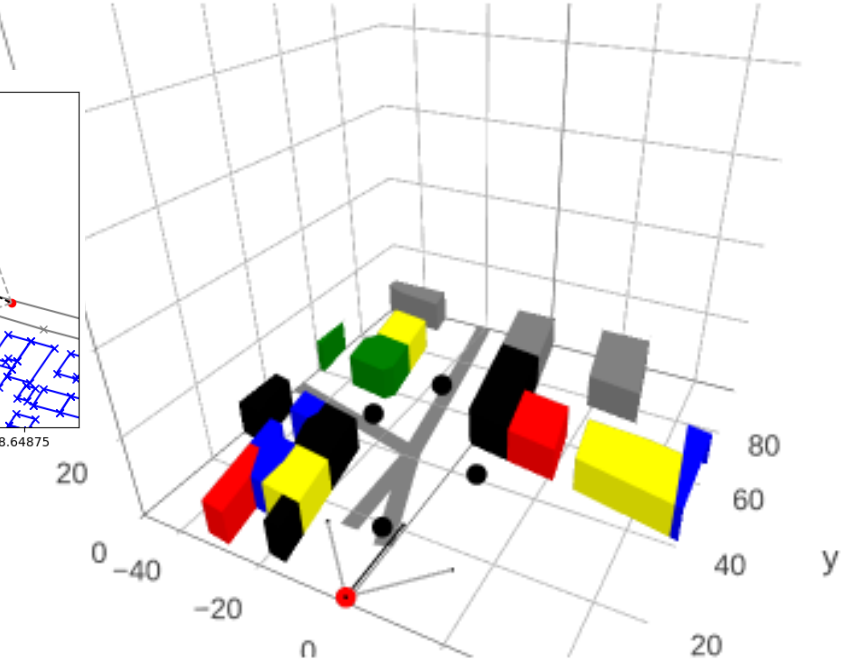
Why build Simulations?

- Possible source of (unlimited) situation examples – Optical flow

Why build Simulations?

- Possible source of (unlimited) situation examples – 2D/3D Simulations from map data



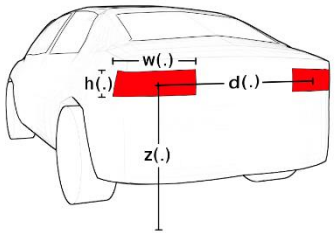


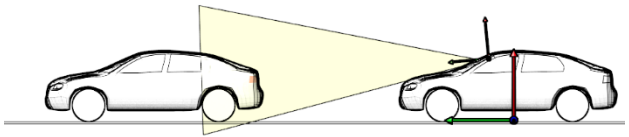
Examples
Brakelight

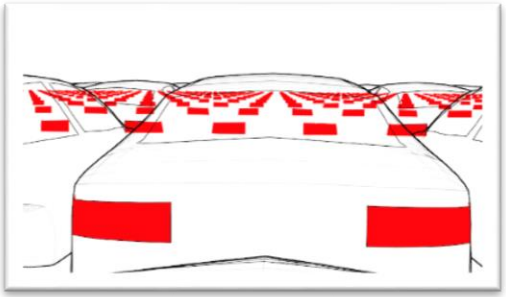
- Task
  - Detect motion anomalies -> simulate ``normal“ image motions
- Parameters
  - Θ_W
    - Geometric: CAD-desc of automotive scenes (cars, streets, buildings, etc.)
    - Photometric: textures, reflectances, external lighting (context)
  - Θ_P
    - Object locations (buildings on side of street, cars on street, etc.)
    - Governed by gravity (everything on ground-plane)
    - Object motions (other cars, people)
  - Θ_P
    - Sensor resolution, possible exposure times, noise characteristics
    - Extrinsic (position,angle) and intrinsic (focal length, central point, distortion) parameters

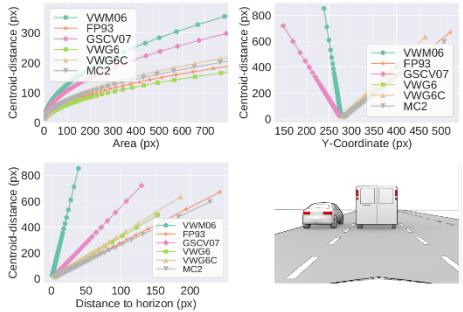
Why build Simulations?

- Possible source of (unlimited) situation examples – 2D projections









Examples
Brakelight

- Task
  - Calculate likelihood of detected blob-pairs (only geometric)
- Parameters
  - Θ_W
    - Geometric: CAD-desc of cars
  - Θ_P
    - Object locations (other cars)
    - Governed by gravity (everything on ground-plane)
  - Θ_S
    - Sensor resolution
    - Extrinsic (position,angle) and intrinsic (focal length, central point, distortion) parameters

Why build Simulations?

- Possible source of (unlimited) situation examples – Photorealistic RGB images

Labels for each patch

Why build Simulations?

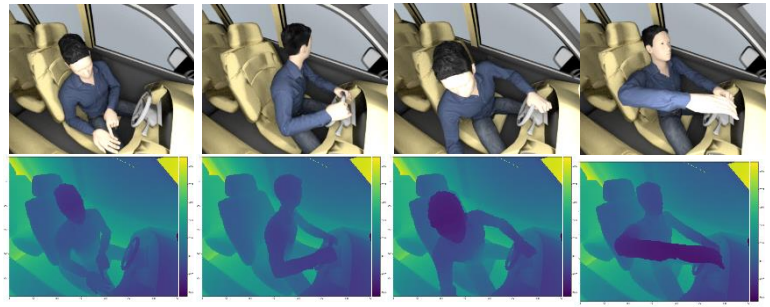
- Possible source of (unlimited) situation examples – Photorealistic RGB images

Examples
Trafficsigns

- Task
  - Detect and recognize traffic signs
- Parameters
  - Θ_W
    - Geometric: CAD-desc of automotive scenes (traffic signs, streets, buildings, etc.)
    - Photometric: textures, reflectances, external lighting (context)
  - Θ_P
    - Object locations (traffic-signs on poles, buildings on side of street, cars on street, etc.)
    - Governed by gravity (everything on ground-plane)
  - Θ_P
    - Sensor resolution, possible exposure times, noise characteristics
    - Extrinsic (position,angle) and intrinsic (focal length, central point, distortion) parameters

Why build Simulations?

- Possible source of (unlimited) situation examples – Depth data



Translation to 3D Simulation Engines

- What is your scene made up of?

Geometric Parameters	Photometric Parameters	Process Parameters
Objects (Meshes) and Bounds	Material	Object poses
	Lighting	Object relations
		Temporal aspects (movements, speeds)

Is Simulation Always Helpful?

- Does your simulation require very accurate complex physics? (Simulating Fluids / Simulating Infrared)
- Does your scene require extensive diversity? (Facial Expressions)

Software Workflow

1. Engines: RealTime VS. Raytrace
2. Scene Content
3. Simulation
    1. Setting Up The Environment
    2. Coding
    3. Rendering
    4. Capturing Segmentation

Engines: RealTime VS. Raytrace

Raytrace

- Highest photorealism
- High render time

RealTime

- Moderate photorealism
- Rendering up to 120 frames per second

Engines: RealTime VS. Raytrace

Raytrace

RealTime

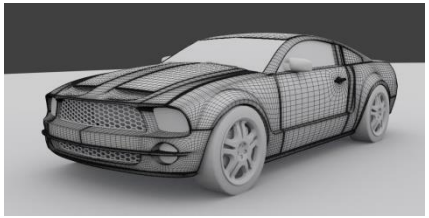
Scene Contents

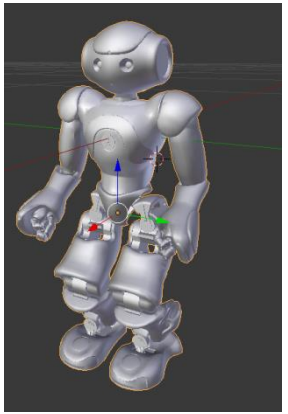
1. Virtual Environment
    1. Models, Material, Animation
    2. Lighting
    3. Process for model placement
    4. Actor behaviour
2. Observer (Camera)
    1. Camera Model
    2. (Post Processing)

Real Venue

Simulated Venue

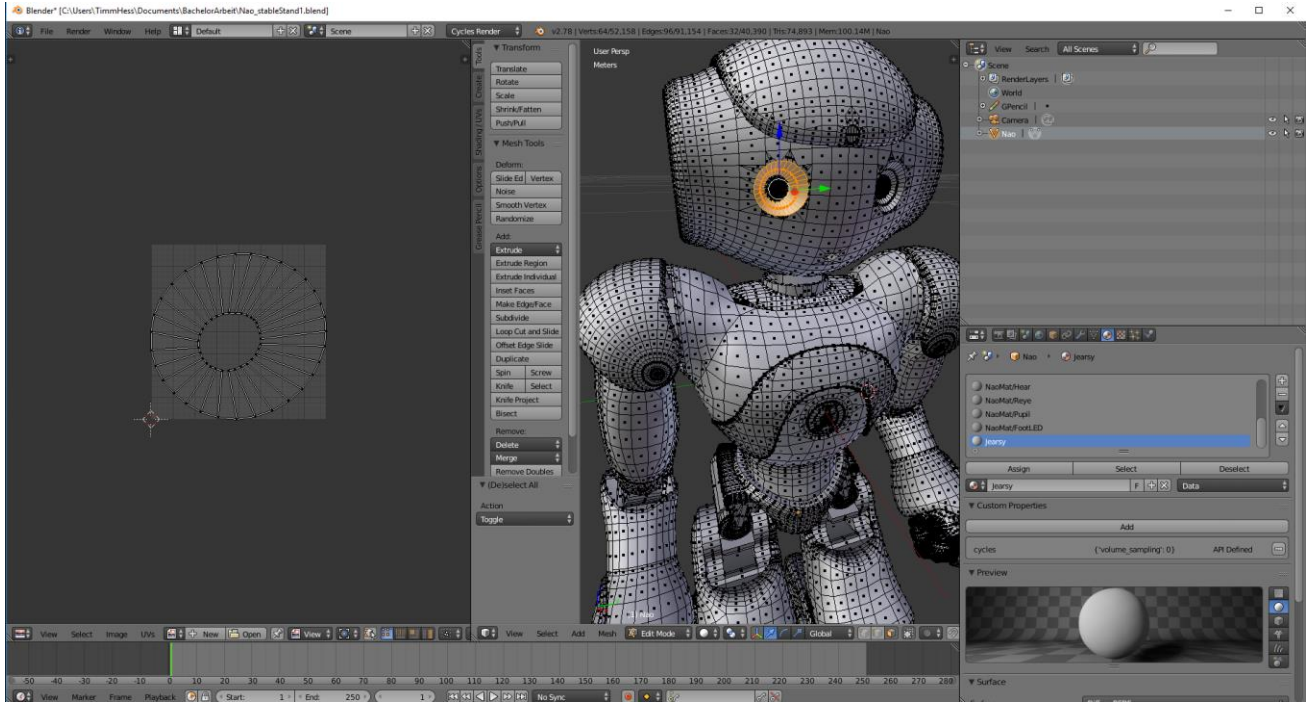
Models



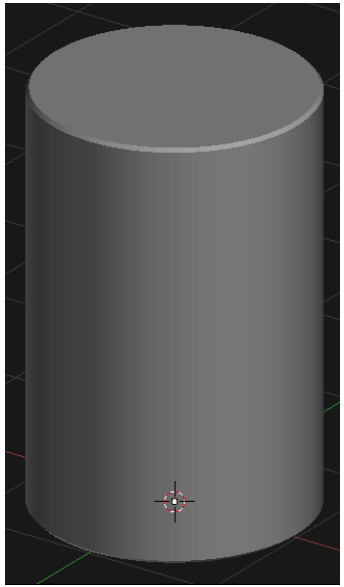


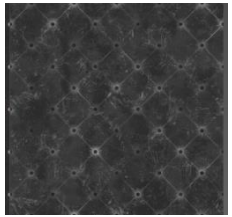
- Modelling
  - Designing 3D Meshes
  - Making texture placement available on the mesh (UV-Unwrap)
  - Animation
- Alternative source: Online Repositories (BlendSwap, Blendermarket, …)

Applying Texture
UV Unwrap

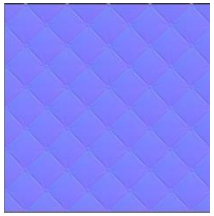


Texture / Material

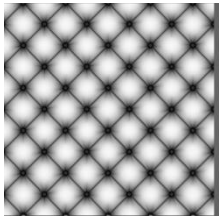




Albedo

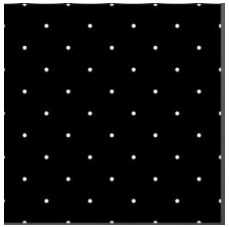


Normal



Height

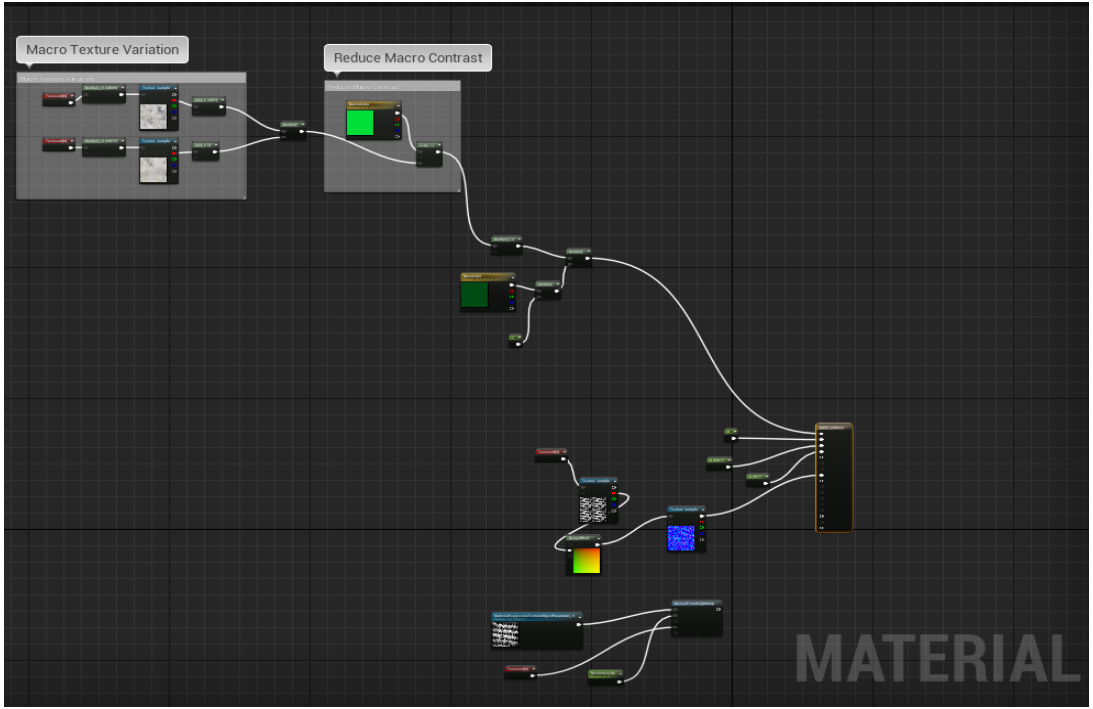
Roughness

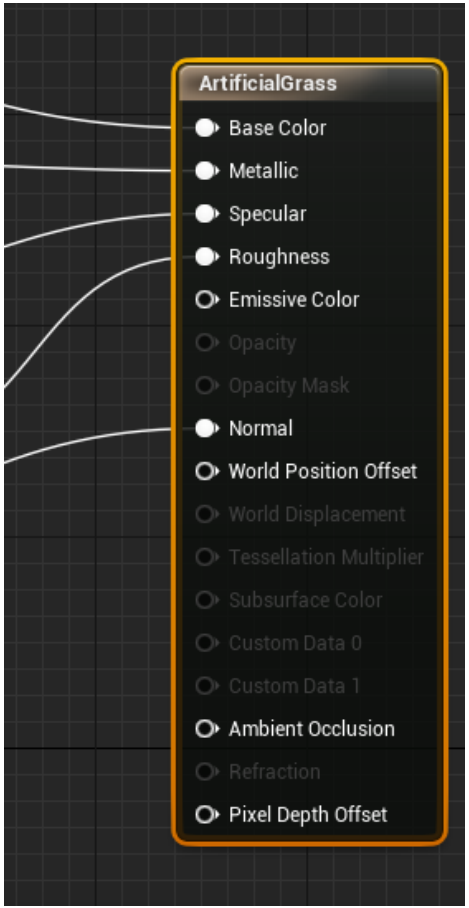


Metallic

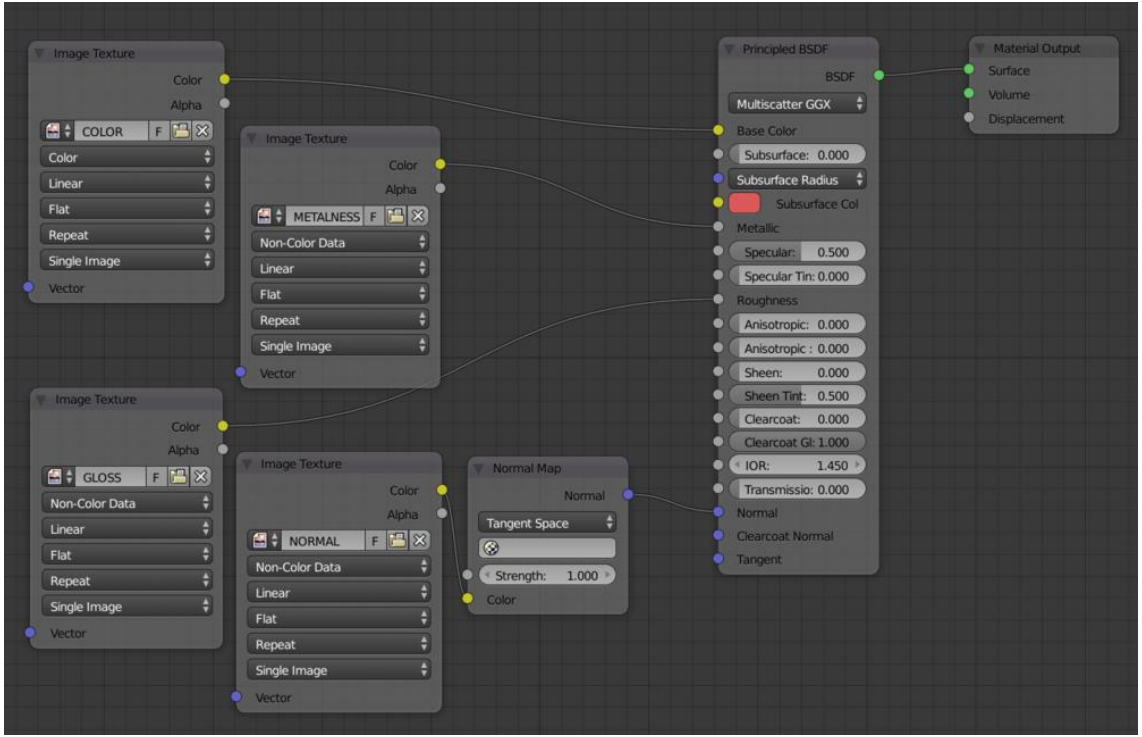
SUBSTANCE

Material (PBR)
UE4





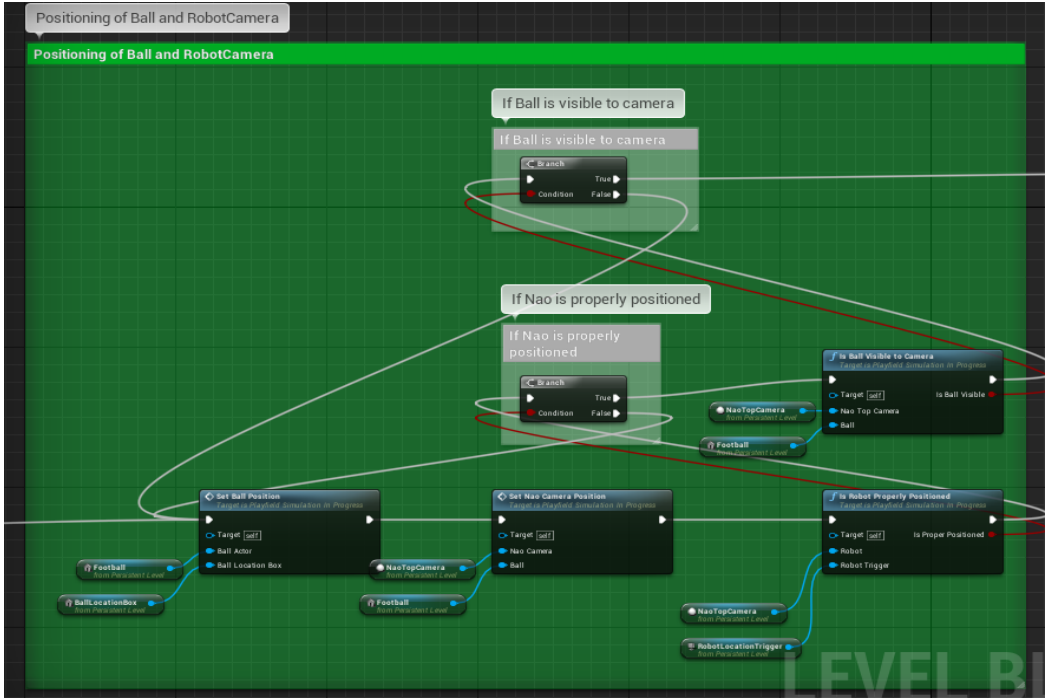
Material (PBR)
Blender



Simulation

Coding

- Set software mechanisms and procedures
  - Object Placement (Stochastic Scene Generation)
  - Lighting Changes
  - Script Activations (Rendering, Segmentation)



Simulation
Rendering

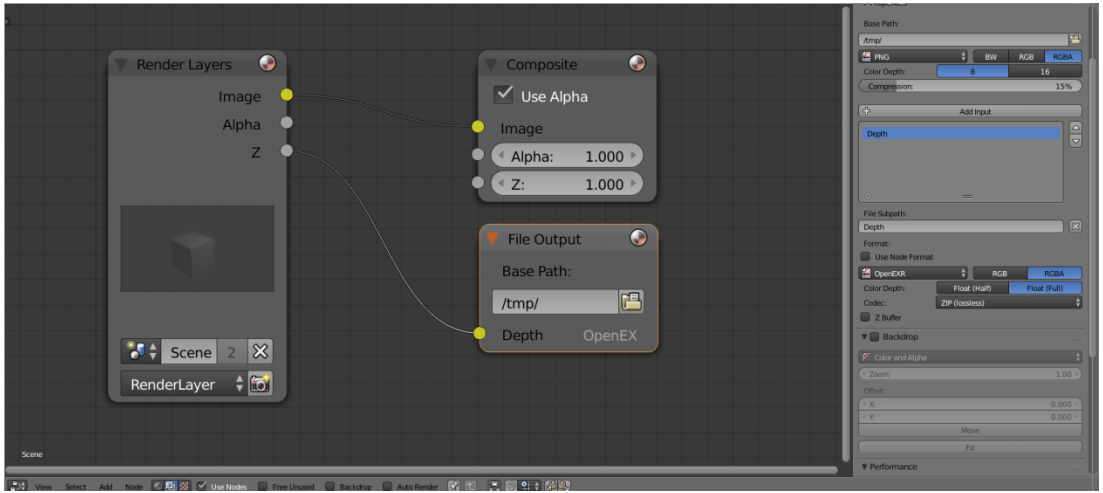
- Pixel Buffer Access
  - Inherit UECameraComponent to gain access to pixel buffers
  - Non trivial due to blackbox-ish rendering pipeline
  - Fast
- Screenshots
  - Screenshot process is asynchronous! Need to freeze frames
  - Slow

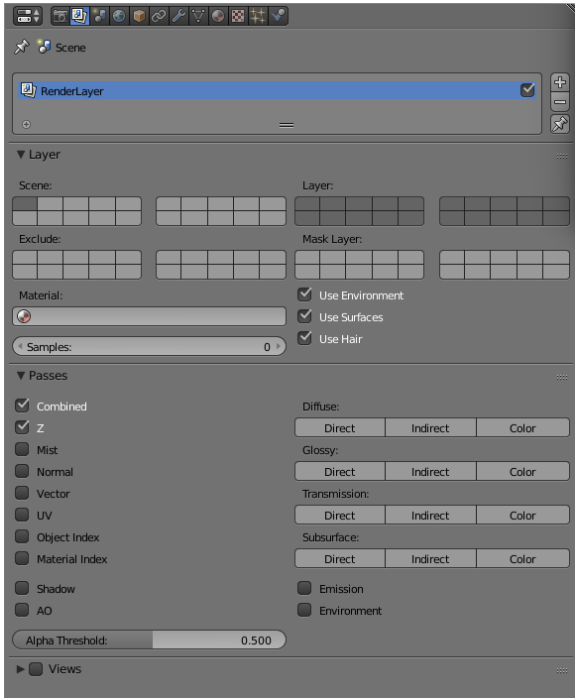
Simulation
Capture Segmentation

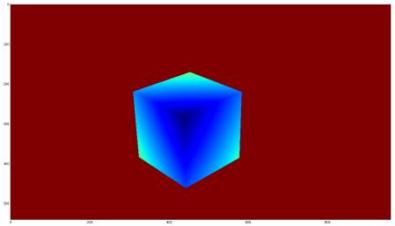
- Shader
  - Propagating render pipeline properties to a shader to access it
    (Multi Condition: Depth, Optical Flow, Objects)
- Raytracing
  - Sending rays through all pixels in the camera viewport and listen to the first object hit by the ray (Slow and objects only)
- Post Processing Material
  - Exploiting a engine build-in post process shader highlighting previously tagged objects (Fast, but error prone due to engine internals such as Antialising! Objects only)

Capture Segmentation

Renderpass - Blender



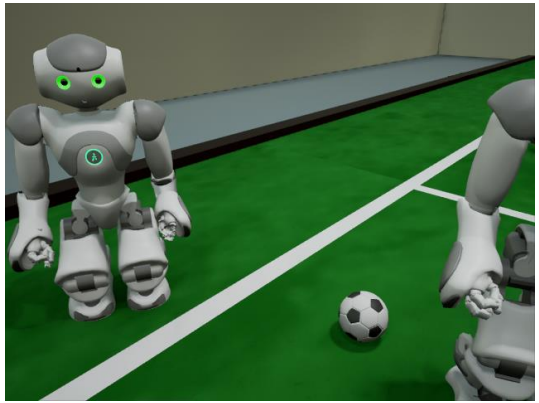


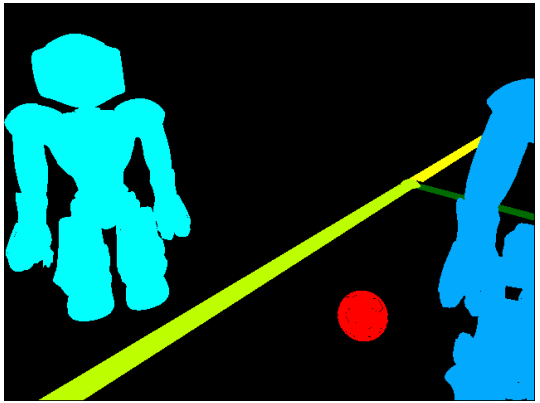


Capture Segmentation

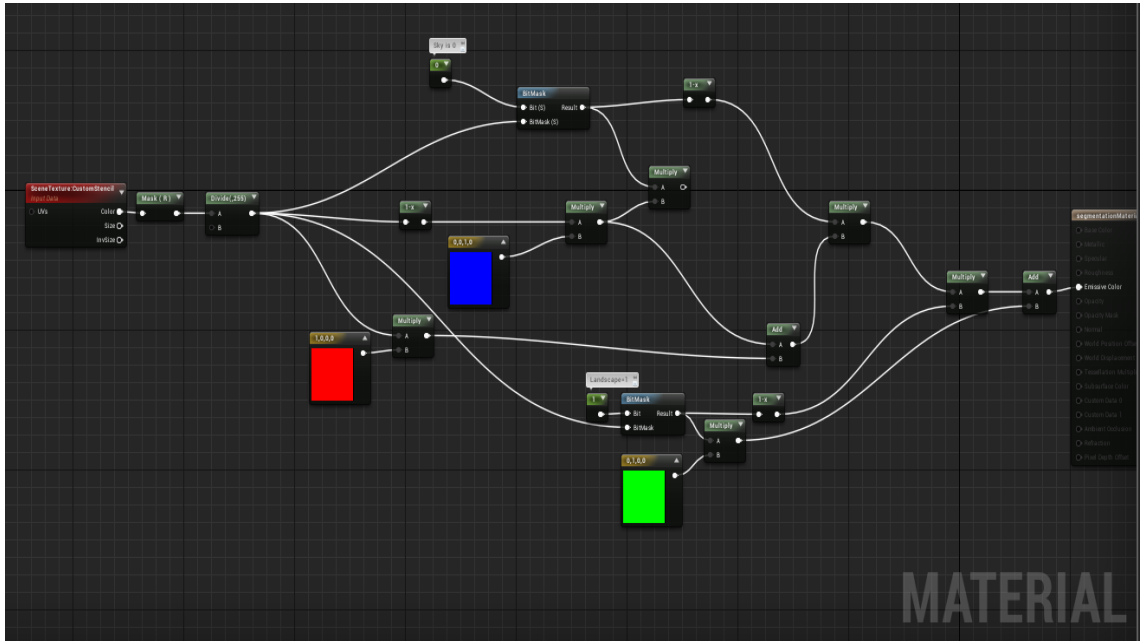
Raytrace – UE4

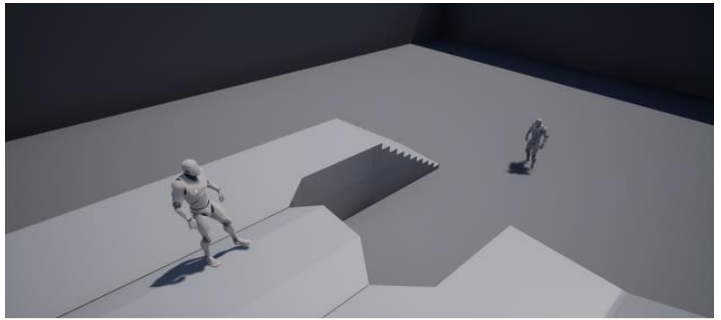
```
for (int y = 0; y < sizeY; y += stride) {
    for (int x = 0; x < sizeX; x += stride) {
        FVector2D ScreenPosition(x, y);
        FVector WorldOrigin, WorldDirection;

        DeprojectScreenToWorld(Player, ScreenPosition, WorldOrigin, WorldDirection);


        // Cast ray from pixel
        bool bHit = World->LineTraceSingleByChannel(HitResult, WorldOrigin, WorldOrigin + WorldDirection * HitResultTraceDistance, TraceChannel, CollisionQueryParams);

        AActor* Actor = NULL;
        if (bHit) {
            Actor = HitResult.GetActor();
            if (Actor != NULL) {
                bool found = false;
                for (int32 i = 0; i < nObjects; i++) {
                    if (objects[i] == Actor) {
                        FString IntAsString = FString::FromInt(i + 1);
                        outputStringMask += IntAsString + " ";
                        counter++;
                        found = true;
                        break;
                    }
                }
                if (!found) {
                    outputStringMask += "0 ";
                }
            }
            else {
                outputStringMask += "0 ";
            }
        }
        else {
            outputStringMask += "0 ";
        }
    }
    outputStringMask = outputStringMask + "\n";
```





Capture Segmentation
Post Process Shader- UE4



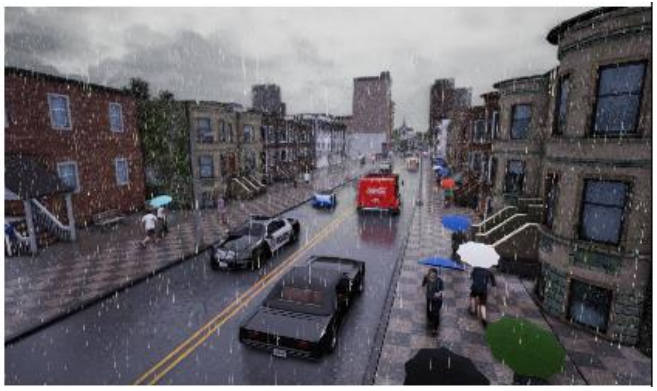


Existing simulation frameworks

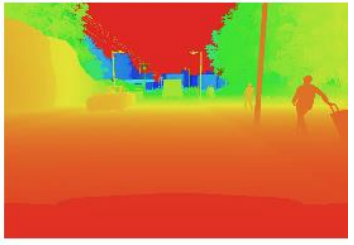
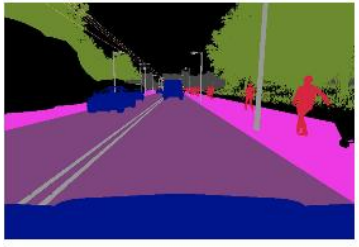
- Usually provide means to
  - User-control objects (cars, drones, etc.)
  - Control environment, weather, illumination
  - Extract most common modalities
    - Rendered images
    - Depth-maps
    - Semantic segmentation
  - Some provide ``game-AI“ (e.g. cars, NPCs)

OpenSource: Microsoft - AirSim

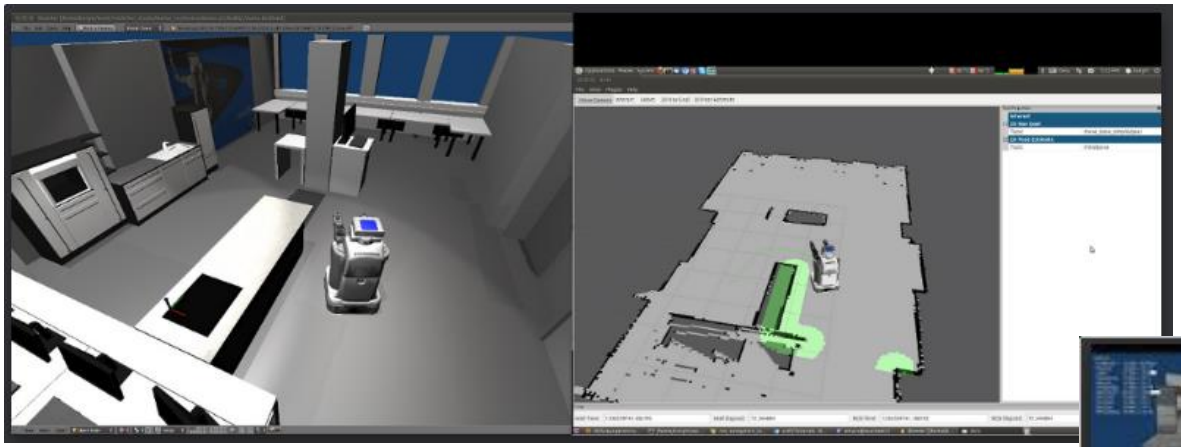
OpenSource: CARLA (2017)



OpenSource: Morse Robot Sim



GTA – Playing for data

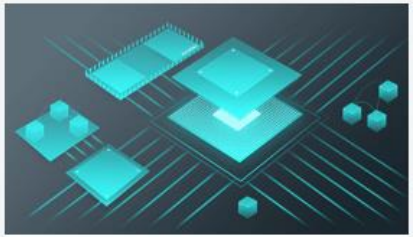
NVIDIA (2018)

Zoox (2017/18)

Baidu – Apollo Simulator

Scenarios

The simulation platform allows users to input different road types, obstacles, driving plans, and traffic light states.

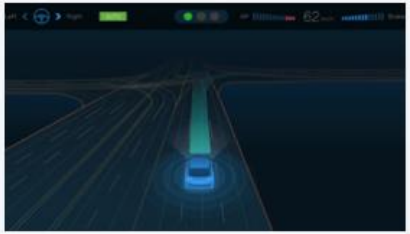


Execution Modes

Gives users a complete setup to run multiple scenarios, and upload and verify modules in the Apollo environment.

Automatic Grading System

The current Automatic Grading System tests via ten metrics: Collision detection, Traffic light recognition and logic, Speed limit, Detection of objects out of lane, End-of-route logic etc.



3D Visualization

Illustrates real-time road conditions and visualizes module output, while showing the status of the autonomous vehicle.

dSpace

And many many others

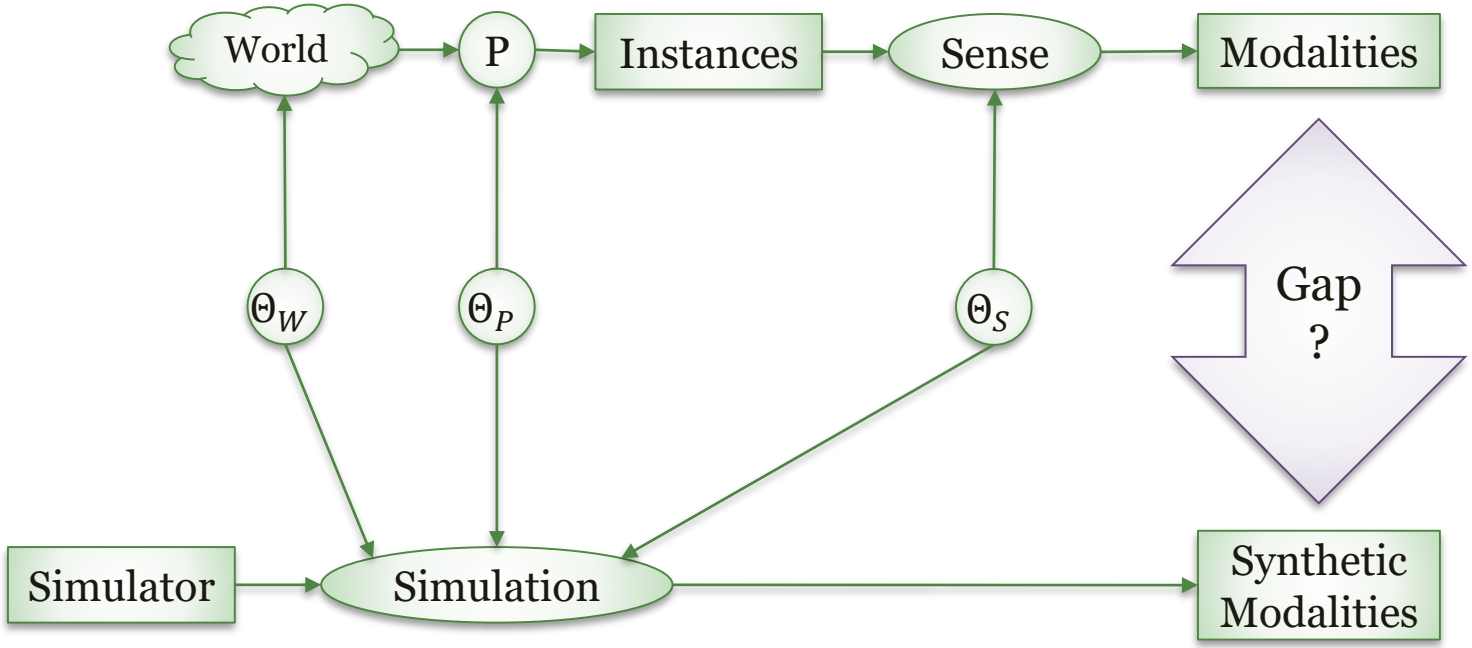
Just to list a few more:

- Uber, Waimo, Daimler, Siemens (Tass), AutonoVi-Sim, VIRES, rFpro, Cogna, SynCity,
- Simulated datasets: Virtual KITTI, Synthia

Evaluating Simulations

- Does your simulation fit your case?
  - Matching of color spaces
  - Matching of corner case conditions (i.e. extreme lighting)
  - How much of your problem space is covered by the simulation?

Evaluating Simulations



Training on Simulated Data
Advantages / Disadvantages

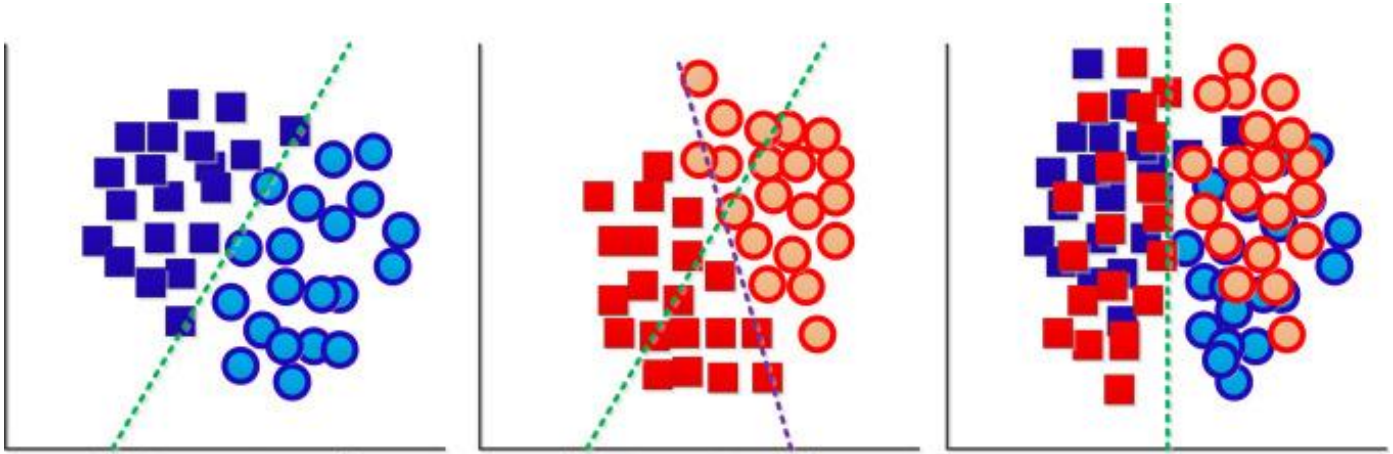
Pro

- Source of (unlimited) labeled data without human annotation efford
- Full control over the environment

Con

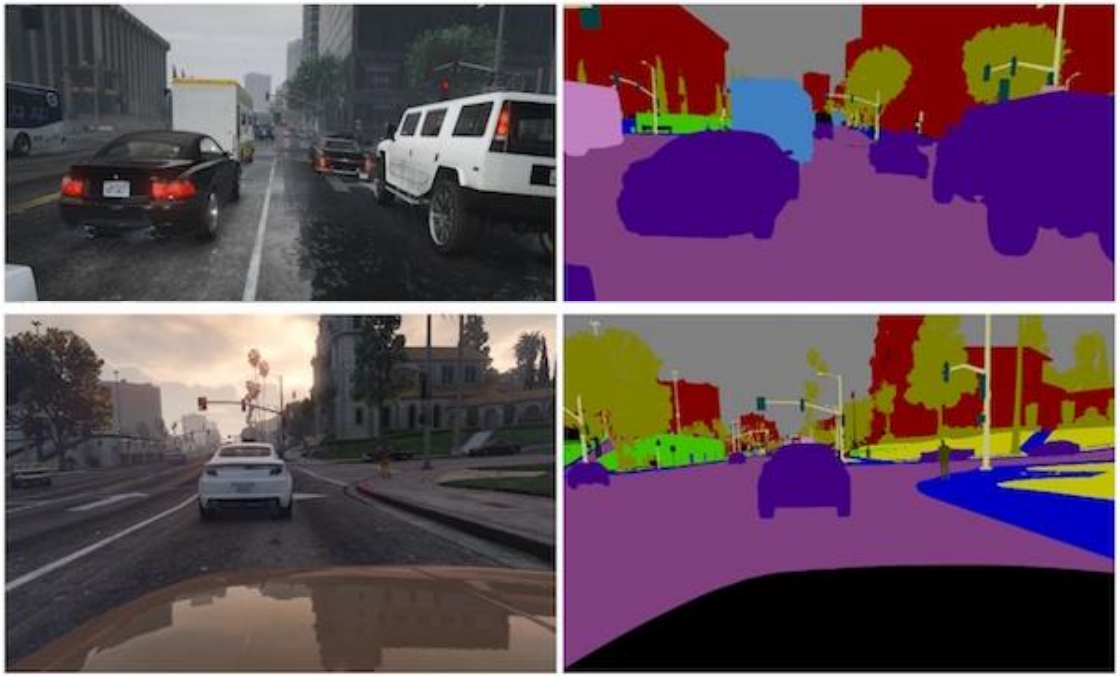
- Obvious shifts in environment from simulated to real and subsequently mismatches in learned statistics when applied to the real world

Domain Adaptation
Combating Domain Shift



Domain Adaptation

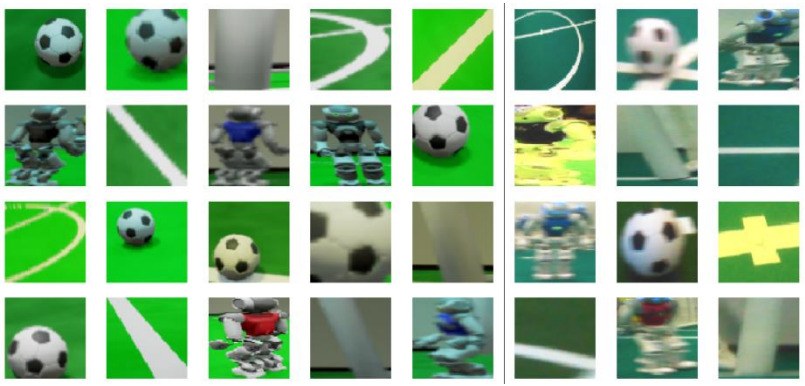
“Rendering Fidelity”



Playing for Data: https://download.visinf.tu-darmstadt.de/data/from_games/

Domain Adaptation
Fine Tuning

- Use additional (possibly human) labeled data from the given target domain to fine tune your model towards target space conditions



Domain Adaptation

GAN – Generative Adversarial Network

winter Yosemite → summer Yosemite

summer Yosemite → winter Yosemite

Domain Adaptation
Other Methods:

Adversarial Methods

- Exploiting Local Feature Patterns for Unsupervised Domain Adaptation [AAAI2019]
- Domain Confusion with Self Ensembling for Unsupervised Adaptation [arXiv 10 Oct 2018]
- Improving Adversarial Discriminative Domain Adaptation [arXiv 10 Sep 2018]
- M-ADDA: Unsupervised Domain Adaptation with Deep Metric Learning [arXiv 6 Jul 2018] [Pytorch(official)]
- Augmented Cyclic Adversarial Learning for Domain Adaptation [arXiv 1 Jul 2018]
- Factorized Adversarial Networks for Unsupervised Domain Adaptation [arXiv 4 Jun 2018]
- DiDA: Disentangled Synthesis for Domain Adaptation [arXiv 21 May 2018]
- Unsupervised Domain Adaptation with Adversarial Residual Transform Networks [arXiv 25 Apr 2018]
- Simple Domain Adaptation with Class Prediction Uncertainty Alignment [arXiv 12 Apr 2018]
- Causal Generative Domain Adaptation Networks [arXiv 28 Jun 2018]
- Conditional Adversarial Domain Adaptation [arXiv 10 Feb 2018]
- Deep Adversarial Attention Alignment for Unsupervised Domain Adaptation: the Benefit of Target Expectation Maximization [ECCV2018]
- Learning Semantic Representations for Unsupervised Domain Adaptation [ICML2018] [TensorFlow(Official)]
- CyCADA: Cycle-Consistent Adversarial Domain Adaptation [ICML2018] [Pytorch(official)]
- From source to target and back: Symmetric Bi-Directional Adaptive GAN [CVPR2018] [Keras(Official)] [Pytorch]
- Detach and Adapt: Learning Cross-Domain Disentangled Deep Representation [CVPR2018]
- Maximum Classifier Discrepancy for Unsupervised Domain Adaptation [CVPR2018] [Pytorch(Official)]
- Domain Generalization with Adversarial Feature Learning [CVPR2018]
- Adversarial Feature Augmentation for Unsupervised Domain Adaptation [CVPR2018] [TensorFlow(Official)]
- Duplex Generative Adversarial Network for Unsupervised Domain Adaptation [CVPR2018] [Pytorch(Official)]
- Generate To Adapt: Aligning Domains using Generative Adversarial Networks [CVPR2018] [Pytorch(Official)]
- Image to Image Translation for Domain Adaptation [CVPR2018]