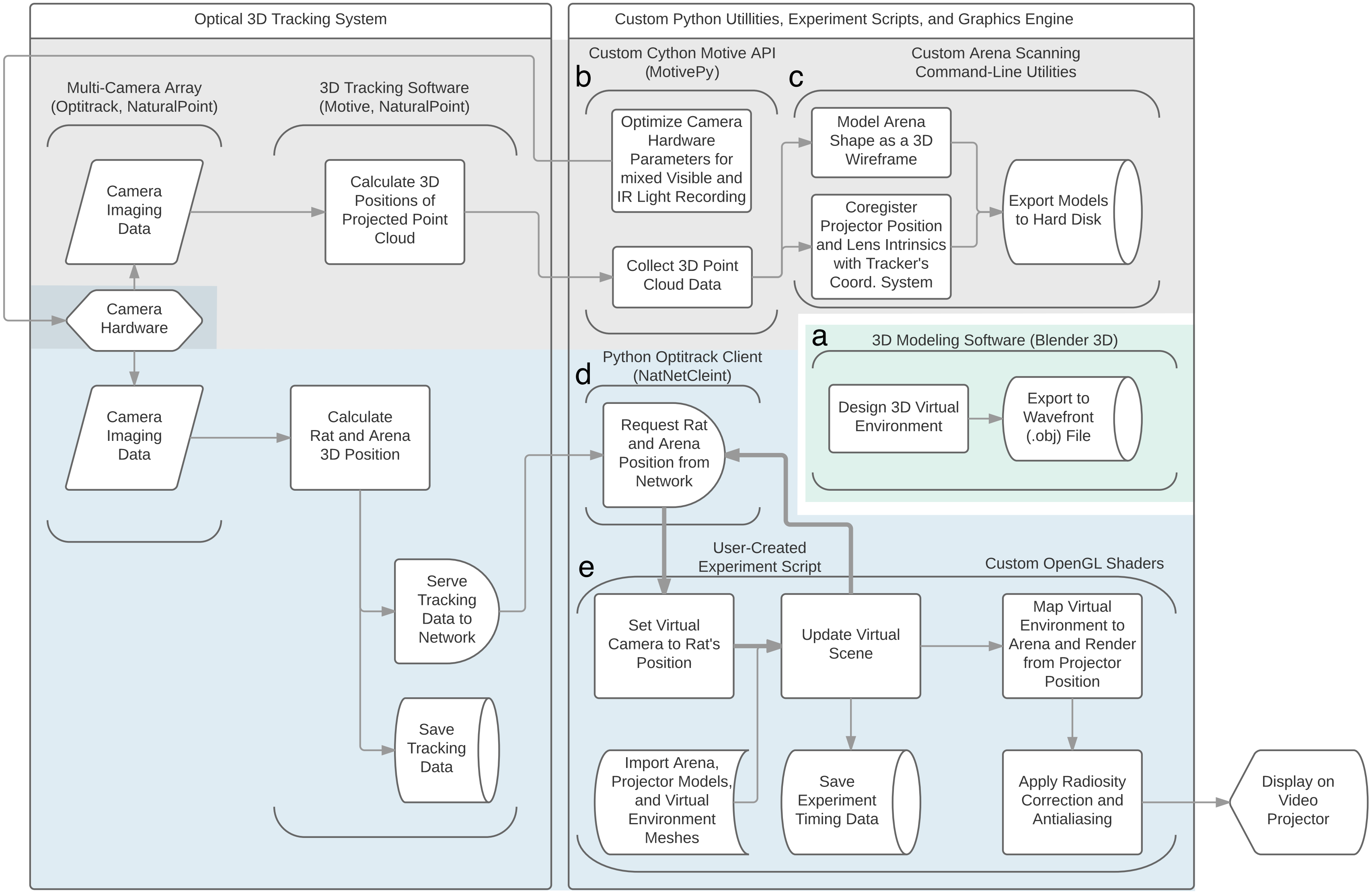
Optical 3D Tracking System
Custom Python Utillities, Experiment Scripts, and Graphics Engine
Multi-Camera Array (Optitrack, NaturalPoint)
3D Tracking Software (Motive, NaturalPoint)
Camera Imaging Data
Calculate 3D Positions of Projected Point Cloud
Camera Hardware
Camera Imaging Data
Calculate Rat and Arena 3D Position
Serve Tracking Data to Network
Save Tracking Data
Custom Cython Motive API (MotivePy)
b
Optimize Camera Hardware Parameters for mixed Visible and IR Light Recording
Collect 3D Point Cloud Data
Custom Arena Scanning Command-Line Utilities
c
Model Arena Shape as a 3D Wireframe
Coregister Projector Position and Lens Intrinsics with Tracker's Coord. System
Export Models to Hard Disk
a
3D Modeling Software (Blender 3D)
Design 3D Virtual Environment
Export to Wavefront (.obj) File
Python Optitrack Client (NatNetCleint)
d
Request Rat and Arena Position from Network
e
User-Created Experiment Script
Custom OpenGL Shaders
Set Virtual Camera to Rat's Position
Update Virtual Scene
Map Virtual Environment to Arena and Render from Projector Position
Import Arena, Projector Models, and Virtual Environment Meshes
Save Experiment Timing Data
Apply Radiosity Correction and Antialiasing
Display on Video Projector

Supplementary Figure 3. RatCAVE hardware-software components flowchart. Each component, depicted as vertical parenthesis, takes information from one source and sends information to another source; information flow is depicted in direction of arrows. Detailed operations of each component are depicted as blocks, and software components are labeled by letter. (**a**) Blender 3D. The virtual environment is created before the experiment using 3D modeling software (right-center module) for loading into the VR experiment script. (**Gray Zone**) Tracking and Setup Coregistration. A Multi-camera array sends imaging data of the rodent's position on each camera to 3D tracking software, which combines the data from each camera image into a single 3D location and sends that position to the ratCAVE environment (left, “Optical 3D Tracking System”). (**b**) MotivePy. The cameras' settings can be modified directly in a Python environment to make visible-light collection possible, a necessary step for arena scanning and projector calibration. (**c**) ratcave_calibrate. Two command-line programs are used for arena scanning and projector calibration. The arena scanning program projects a moving grid of white dots on the arena surface, collects the 3D positions of the projected points via the camera array, and fits the resultant point cloud to a 3D mesh model of the arena. The projector calibration program maps single points displayed from the projector onto the 3D position of the arena, one at a time. It then uses OpenCV's camera_calibrate tool to use these mappings to find the position of the projector in the camera array's coordinate space. (**Blue Zone**) VR Engine. (**d**) NatNetClient. Rat position data is collected in real-time from the camera array and brought into the Python environment, for use in VR experiment scripts. (**e**) Fruitloop. The virtual environment (VE) is rendered in a Python 3D graphics engine. The VE is loaded from file (created in Blender 3D), and on each display frame, using the rodent position data to move a virtual camera to the rodent's position in a virtual environment (blue zone). This process, encompassing the core of the VR engine, (get rodent position, move camera, update and render scene) occurs in a loop, repeated each frame, with the frames themselves sent to the GPU for arena mapping and shading (examples on Supp. Figure 1d) and then to the video projector (bottom-right corner). See the “Software” section in Online Methods for more details.