Advanced Real-Time Radar Simulation: Maintaining Range Resolution for Large-Area Ground Maps

Radu Visina, Colton Smith, Greg Carter, David Kirk
Information Systems Laboratories
San Diego, CA
rvisina@islinc.com, csmith@islinc.com, gcarter@islinc.com, dkirk@islinc.com

ABSTRACT

Modern advances in Graphics Processing Units (GPU) technology and the science of 3D computer graphics have stirred interest in the use of these platforms and algorithms to perform real-time simulation of radar antennas, radar signal processing techniques, and radar data output. Applications include aircrew and sensor operator training systems, knowledge-aided clutter suppression, hardware-in-the-loop testing, and gaming. Often, the type of data to be simulated is the receiver signal amplitude as a function of range and azimuth angle (or sometimes range and Doppler shift). Such data is required for generating radar images, such as SAR or Real Beam, and for performing detection routines for use in tracking algorithms. However, the GPU pipeline technique (including rasterization and/or ray tracing) presents a technical challenge: the focal plane camera approximation of a 3D graphics "receiver" senses the scene in a perspective projection space, while radar senses the scene in a polar or spherical space. The nonlinear transformation from the perspective projection space to a polar space decreases range resolution at longer ranges, while it is known that true radar can maintain nearly constant range resolution throughout its range swath. This undesired effect is amplified under the conditions of large azimuth and/or large elevation coverage of the camera's view frustum in large-area scenes such as Real Beam images from monostatic radar antennas. This paper mathematically explains the problem and develops a novel solution that is then numerically demonstrated through simulation: the oblique camera frustum. The technical breakthrough presented in this paper is demonstrated using the Unity simulation platform. The work was developed under a Small Business Innovation Research (SBIR) Phase II project awarded at the inaugural Air Force Sim Pitch Day event at I/ITSEC 2019.

ABOUT THE AUTHORS

Radu Visina (PhD EE University of Connecticut) completed the Ph.D. in Electrical Engineering from the University of Connecticut. He is a research engineer at ISL and has been actively developing theory and practical solutions for advanced radar target tracking and research into high-fidelity, model-based, GPU-accelerated real-time radar simulation. In publications, Dr. Visina has advanced the theory and practical methods of tracking maneuvering targets, including highly-maneuvering/goal-oriented targets, and the track-to-track fusion of maneuvering target state estimates.

Colton Smith is a Software Engineer at ISL and current master's student seeking a Master of Science Degree in Computer Science at Ohio University, and holds a Bachelor of Science in Computer Science. His research focuses on improving more bio-feasible learning methods for neural networks, and his work is primarily development within the Unity engine.

Greg Carter (BS, Systems Engineering, Wright State University, MS, Aviation Systems, University of Tennessee Space Institute, Graduate). Vice President of ISL Dayton Operations and Program Manager with over 35 years of experience in Industry and DoD acquisition and operations. He is a Federal Aviation Administration Commercial Pilot and an Air Force Command Pilot and has operational, instructor pilot, and Flight Examiner experience in the F-111. He is a graduate of the U.S. Naval Test Pilot School and was an F-16 Experimental Test Pilot and Instructor Pilot.

David Kirk (BS, Physics, Texas A&M University, MS, Physics, The Pennsylvania State University), a Vice President and Principal Engineer with ISL, has over 30 years of experience in systems analysis and modeling of sensor systems, specializing in the areas of FOPEN synthetic aperture radar (SAR), automatic target recognition (ATR) in both radar and hyperspectral imagery, ground moving target indication (GMTI) radar, ground target tracking, exploitation of full motion video, and navigation/geolocation accuracy.

Advanced Real-Time Radar Simulation: Maintaining Range Resolution for Large-Area Ground Maps

Radu Visina, Colton Smith
Information Systems Laboratories
San Diego, CA
rvisina@islinc.com, csmith@islinc.com

Greg Carter, David Kirk
Information Systems Laboratories
San Diego, CA
gcarter@islinc.com, dkirk@islinc.com

INTRODUCTION

Computer-aided radar simulation platforms have existed for some time. Applications include aircrew and sensor operator training systems, knowledge-aided clutter suppression, hardware-in-the-loop testing, and gaming. The focus of this paper is specifically on large-area ground map imaging radar whose image data are mapped in one axis from angle (azimuth angle specifically) or Doppler shift, and in the second axis from range. Two primary examples of such images are Real Beam (RB) and Synthetic Aperture Radar (SAR). While SAR is primarily used for imaging, RB data is also used for automated aircraft Terrain Following (TF) feedback control systems. Data for SAR is collected in the Doppler/range dimensions while for RB it is collected in azimuth/range dimensions.

The approach to radar image simulation used in this paper first describes the scene using 3D mesh data, which can include terrain and objects for cultural features, with possible texture skins on any or all meshes. This is standard for modern gaming and high-fidelity simulators. Second, the approach samples the desired world data using a virtual perspective projection camera, which is a process inherent in most modern gaming and simulation engines and solves the visibility problem with computational efficiency. In this step, the simulation can utilize modern screen-space ray-tracing techniques without loss of generality; see (Hossain et al., 2020) for this approach in radar simulation. Third, it simulates complex-valued radar receiver channel data, sometimes referred to as in-phase & quadrature (I&Q) data, from a scene sampled by the virtual perspective projection camera. In the fourth step, much like in a true radar image generation system, the magnitude of the I&Q data is projected onto a Cartesian ground plane for visualization. The advantage of this approach is that natural radar image phenomena, including shadowing, layover, geometric skewing, and speckle are simulated with no additional simulation steps, and the primary, raw I&Q data generation can be used for applications other than image simulation, such as the evaluation of signal processing algorithms. The main challenge is that a histogram binning process is required for the generation of I&Q data. This process is a change of space from the camera's perspective space to the radar's polar space, and this change of space can result in a loss of fidelity/information when the range and/or angle extents of the data are large. This paper develops a method to significantly reduce the problem while maintaining all the advantages of the I&Q data generation approach using perspective camera sampling.

The technique to maintaining range resolution that is developed in this paper relies on the creation of a *horizontally-aligned, oblique camera frustum*, as opposed to the more conventional *symmetric camera frustum* that software developers and users are familiar with. See Figure 1 for examples of radar images simulated using both approaches and see Figure 2 for visual examples of the conventional and proposed frustum types. Readers may be familiar with the *lens shift* feature of photographic cameras, which is often used to capture optical photographs of tall architectural structures from a ground position. Using lens shift, the dimensions of the structure appear consistent throughout the height of the photograph, and the vertical features of building-like structures remain parallel. In the mathematics of 3D graphics, a virtual lens shift is equivalent to creating an oblique camera frustum.

Literature on the topic of radar simulation has grown in recent years and there are different approaches for large area simulation (Balster et al., 2017), (Lin et al., 2016), (Martelli et al., 2019), (Zhang et al., 2013), (Schöffmann et al., 2021). One common approach is to use 3D graphics to simulate just the final output image, such as a SAR image, as is done in (Peinecke et al., 2008), but such approaches are very approximate when it comes to geometric correctness as described in (Balz et al., 2015). According to Balz. et. al., for geometric correctness, it is desirable to first simulate *raw radar data* in polar coordinates, and this is more like what a true radar receiver would present to a data processing and imaging subsystem.

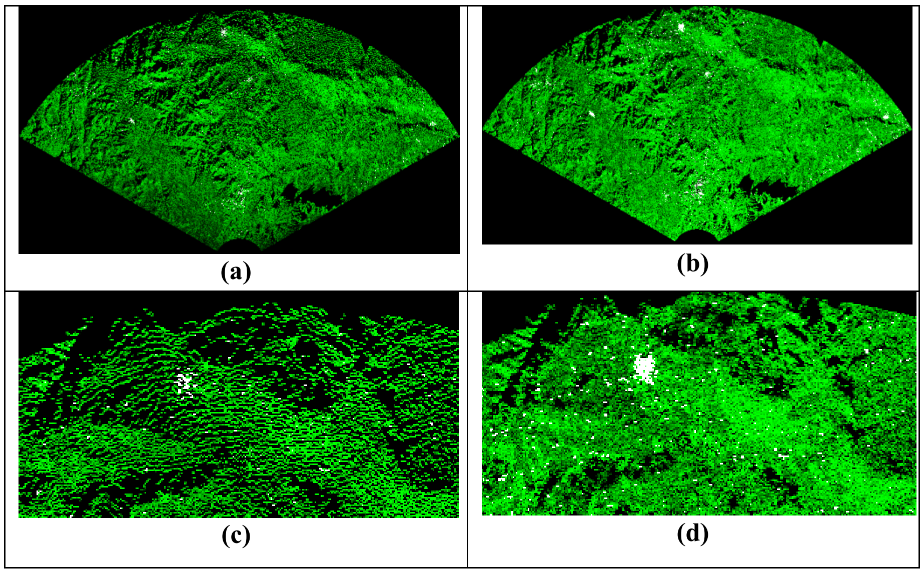


Figure 1: Exemplified Range Resolution Problem. (a) - loss of information data can be seen at far range. (b) Consistent, desired range information. (c) – zoom in of (a) (d) – zoom in of (b)

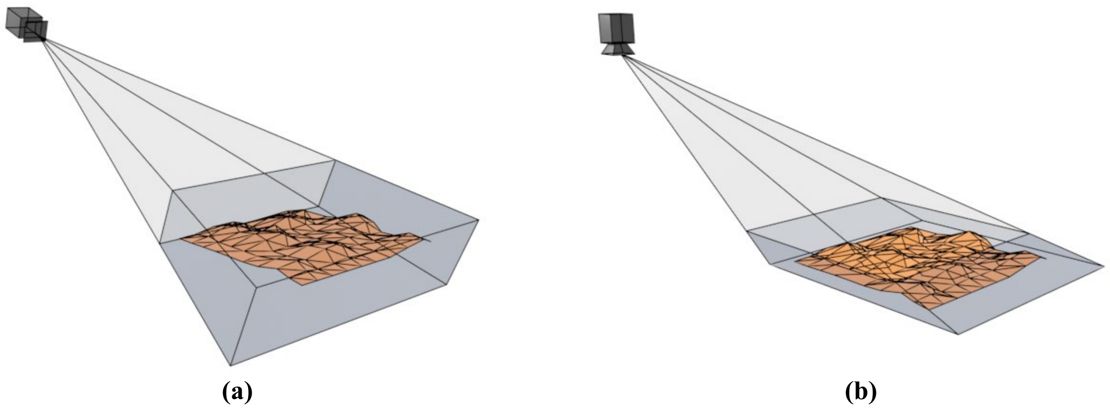


Figure 2: Examples of two virtual camera frustum types capturing a terrain scene. (a) - Symmetric Frustum (b) - Oblique Frustum. Notice that in (b), the camera is pointing downwards, so the frustum is aligned to the horizontal plane.

The approach described here lends itself to the utilization of modern GPUs, shader programs, game engines, and the fast rasterization/z-buffering algorithms that have made modern games and other optical simulations high in fidelity, realistic, and real-time performant. It is natural to extend such systems and algorithms to general radar simulation. While the theory and algorithms described in this paper can be used on any modern GPU hardware, operating system (OS), and game/sim platform, the simulation section will be described in terms of the Unity game engine, owing to its wide and flexible applicability, large user/community base, excellent documentation and tutorials, and wide hardware/OS support.

The organization of this paper is as follows. It will introduce the reader to the use of the 3D graphics rasterization pipeline for radar raw data generation. A discussion of the drawback of using the conventional symmetric frustum follows, with the introduction of the oblique frustum and its advantages. Then, formulas required for configuring both frustum types will be derived, and formulas for a measure of the camera's range sampling resolution for both frustum types will also be derived and plotted over a segment of possible range extents. Finally, the paper will describe the simulation scene that was used to output the images of Figure 1.

SIMULATION TECHNIQUE AND PROBLEM STATEMENT

Terrain and Building Data

All world data that enters the GPU rasterization pipeline must be eventually described as *meshes* with optional *texture skins.* Mesh data consists of two lists: vertices and indices. Vertices contain position and other information for all vertices of 3D objects, and the indices order triangles out of the defined vertices. Textures are two-dimensional bitmap images that are "wrapped over" meshes and provide higher resolution information for objects that would require excessive memory and computational resources to display if described with meshes alone. Texture skins are optional, but highly recommended for encoding high-fidelity world information. Without loss of generality, this paper only will utilize vertex geometry to encode world-space positions. Buildings will also be described in the scene, though for simplicity, no texture skins will be applied; this assumes a single, homogeneous material cover on the building faces.

The Unity generates meshes for the terrain (i.e., the ground, or landscape) from heightmaps. A heightmap is a simple two-dimensional structure of height information on a regular grid that can be stored as a texture and assumes all geometric features are convex, meaning that land features such as caves cannot be described in a heightmap (though they can be superposed on top of the heightmap using meshes). Internally, an engine such as Unity must convert the heightmap into a triangulated mesh for rasterization purposes. Buildings and other objects are directly stored as meshes.

A local world coordinate system will be used in this paper. The world axes x, y, z are defined for as follows: x points east, y points towards the sky (normal to Earth's surface, i.e. altitude above sea level), and z points north. This convention corresponds to the Unity engine's specifications. All 3D points will be denoted using vector notation $\mathbf{s} = [x \quad y \quad z]'$ with $[\,]'$ denoting matrix transpose.

A terrain bounding box is first specified around the terrain of interest with one corner at $\mathbf{s}_{\text{tb}}$ and dimensions W_x, W_y, and W_z depending on the minimum range $r_{\min}$, maximum range $r_{\max}$, minimum bearing angle $\theta_{\min}$, and maximum bearing angle $\theta_{\max}$. In addition, four points, $\mathbf{s}_{\text{close}}$, $\mathbf{s}_{\text{far}}$, $\mathbf{s}_{\text{left}}$, and $\mathbf{s}_{\text{right}}$, are assigned based on the bounding box. See Figure 3 for a description of these parameters.

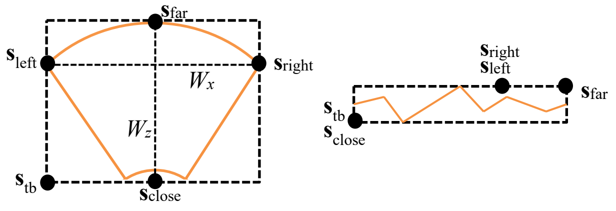

$\mathbf{s}_{\text{far}}$
$\mathbf{s}_{\text{left}}$
$\mathbf{s}_{\text{right}}$
W_x
W_z
$\mathbf{s}_{\text{tb}}$
$\mathbf{s}_{\text{close}}$
$\mathbf{s}_{\text{right}}$
$\mathbf{s}_{\text{left}}$
$\mathbf{s}_{\text{far}}$
$\mathbf{s}_{\text{tb}}$
$\mathbf{s}_{\text{close}}$

Figure 3: Description of terrain bounding box and points for used for configuring camera frustum. Top view (left), side view (right)

The Virtual Radar Antenna

A radar antenna senses the scene in a spherical coordinate system consisting of range r, azimuth angle θ, and elevation angle ϕ. These three coordinates are relative to the antenna's position and orientation. The antenna's position in world space is $\mathbf{s}_{\text{a}}$.

For radar ground maps such as RB, it can be assumed that the antenna is pointing horizontally with no roll, has a wide elevation beamwidth, and has a very narrow azimuth beamwidth.

In ground mapping applications at long standoff range, the two-dimensional image information comes from the radar receiver's azimuth and range information (Richards, 2014). A physical radar's range resolution is based on bandwidth; however, in this paper, the virtual antenna's range resolution is assumed to be consistent with the pixel resolution of the desired image so that information is not blurred in range. For example, if the resulting image captures 10 kilometers of range and displays the image using 1000 rows and columns of pixels, then the simulated radar resolution is assumed to be at most 10 meters. Defining r_{Δ} as the radar resolution distance, r_{sw} as the range swath, and N_{px} as the number of pixel rows and columns in the final image (we assume square output textures for simplicity), this can be stated as

$$r_\Delta \leq \frac{r_{\max} - r_{\min}}{N_{\mathrm{px}}}. \tag{1}$$

In this paper, it is also assumed that the azimuth resolution (i.e., the beamwidth) is also consistent with the displayed image so that the image is not blurred at the farthest extend of range. Similar to (1) and with θ_{sec} the total angle sector of the scan, this can be stated as

$$\theta_\Delta \leq \frac{\theta_{\mathrm{sec}}}{N_{\mathrm{px}}}. \tag{2}$$

In the simulated radar environment described in this paper, the achievable resolutions of the radar system can be arbitrarily small. What is of primary concern is the topic of the paper: achieving the desired resolutions when using a virtual perspective projection camera to sample the scene. These resolutions have a different interpretation and will be described next.

The Camera Frustum, Perspective Projection, and Rasterization

The motivation to use a virtual perspective projection camera for radar simulation is that the rasterization graphics pipeline with perspective projection mathematics solves the classic visibility problem very efficiently (Lengyel, 2012). A reflective point $\mathbf{s}$ in world space is visible to the viewer at camera position $\mathbf{s}_{\mathrm{c}} = \mathbf{s}_{\mathrm{a}}$ if and only if there is a direct line of sight between $\mathbf{s}$ and $\mathbf{s}_{\mathrm{c}}$ (i.e. there are no occlusions). Fortunately, this is true for visibility in a spherical coordinate radar antenna as well.

In the rasterization pipeline, the virtual camera uses the pinhole approximation, where it can be assumed that all rays of information from the scene intersect at the point $\mathbf{s}_{\mathrm{c}}$, where a hypothetically infinitesimal pinhole exists in an otherwise opaque plane. A viewing plane $\mathbf{p}_{\mathrm{v}}$ exists *in front of* the camera[1]. A true physical pinhole camera cannot exist because the light would be fully attenuated as it attempts to pass through the infinitesimal pinhole. Ignoring this attenuation in a simulation is possible, and results in a perspective projection viewing system without depth of field effects (i.e. there is no blurring due to objects being out of focal distance).

The camera *view space* (sometimes called just *camera space*) is defined as a Euclidean space that is translated and rotated from world space and will be referred to by a superscript c in the mathematics. In view space, the camera is located at the origin $\mathbf{s}_{\mathrm{c}}^{\mathrm{c}} = \mathbf{0}$. The standard convention in Unity (coming from DirectX convention, and that which will be used in this paper) is that the z axis points directly in front of the camera, the x axis points from the left of the viewing screen to the right, and the y axis points from the bottom of the screen to the top (a *left-handed projection* orientation). OpenGL convention reverses the direction of the z axis. In camera space, the viewing plane is located at the focal distance d_{v} and has normal vector $\mathbf{v}_{\mathrm{v}}^{\mathrm{c}} = [0 \quad 0 \quad 1]'$.

The virtual camera *samples* the desired scene in a *perspective projection space*, where information is converted from the world space mesh data to square $N_{\mathrm{px}} \times N_{\mathrm{px}}$ textures here termed *perspective projection textures*. Most non-orthographic images of a 3D scene that the reader is familiar with can be considered perspective projection textures displayed directly on a monitor screen. However, with modern GPUs, the functionality can be extended to further extract information from, and to post-process, these textures rather than just displaying them to the viewer, and they can contain high-precision information in their pixels other than just optical colors for display on a monitor. The camera only sees what is within its *view frustum*. The frustum is a rectangular pyramid bisected by a plane called the *near plane* $\mathbf{p}_{\mathrm{n}}$ (see Figure 2). In this paper, it can be assumed that $\mathbf{p}_{\mathrm{n}} = \mathbf{p}_{\mathrm{v}}$ without loss of generality. The GPU *clips* all mesh information outside of this frustum and transforms all vertices within it in such a way that the frustum space is transformed into a unit cube. This transformation is followed by the final rasterization process, where mesh triangles are drawn on the output texture by simply determining which fragments (i.e. pixels) the triangles fall in, and storing the z value (depth) of each triangle's fragments. Visibility is determined by the depth of the fragment – if two separate triangles' fragments fall in the same texture fragment, the fragment with the lower depth value remains and any triangle fragments with higher depth value are discarded. This efficient solution to the visibility problem is often called the *z-buffer algorithm* or *z-testing* and is the primary motivator for using the rasterization pipeline to generate raw radar data achieving real-time computational performance.

[1] In a true pinhole camera, the viewing plane sits behind the pinhole and shows an image of the scene that is flipped vertically and horizontally; by assuming a hypothetical plane is in front of the pinhole, it can be said that the viewer is located at $\mathbf{s}_{\mathrm{c}}$ and the image on the plane is not flipped.

A point in *homogeneous coordinates* will be denoted as

$$\mathbf{t} = \begin{bmatrix} x & y & z & w \end{bmatrix} \tag{3}$$

Representation using homogeneous coordinates allows for translation transformations to be performed as linear matrix-vector operations and simplify the operation of *perspective divide*, which is what shifts vertices with greater depth closer to the center of the cube-transformed frustum before triangle rasterization.

The operations required to transform the world-space position $\mathbf{s}$ of a 3D vertex to the normalized device coordinates are the series of multiplications by the 4×4 view matrix $\mathbf{P}_\text{v}$ and the projection matrix $\mathbf{P}_\text{p}$, where the view matrix consists of the translation and rotation required to bring $\mathbf{t}$ into the camera's view space. The projection matrix can be described as

$$\mathbf{P}_\text{p} = \begin{bmatrix} \frac{2D_\text{n}}{D_\text{r} - D_\text{l}} & 0 & \frac{D_\text{r} + D_\text{l}}{D_\text{r} - D_\text{l}} & 0 \\ 0 & \frac{2D_\text{n}}{D_\text{t} - B} & \frac{D_\text{t} + D_\text{b}}{D_\text{t} - D_\text{b}} & 0 \\ 0 & 0 & -\frac{D_\text{f} + D_\text{n}}{D_\text{f} - D_\text{n}} & -\frac{2D_\text{n} D_\text{f}}{D_\text{f} - D_\text{n}} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{4}$$

where D_n, D_f, D_r, D_l, D_t, and D_b are the locations of the near, far, right, left, top, and bottom planes on the viewing plane, respectively (Lengyel, 2012). Rather than specifying the D_r, D_l, D_t, and D_b parameters directly, the camera frustum will be described by alternate parameters the next sections.

A vertex's homogenous position $\mathbf{t}$ is transformed into the normalized device coordinate $\mathbf{s}_\text{p}$ by first computing the homogeneous point $\mathbf{t}_\text{p} = \mathbf{P}_\text{p}\mathbf{P}_\text{v}\mathbf{t}$ then dividing all of its components by w_p and extracting only the first three components. This division is termed *perspective divide* in computer graphics and is the final step that creates the unit cube from the starting frustum. This operation is performed internally by the GPU rasterization pipeline before entering the *fragment shader* program, where the parallel per-fragment (in the present case, equivalent to per-pixel) computations are performed. What is provided to the fragment shader program are the interpolated values of per-vertex data. The interpolations required per fragment for radar image simulation are world space position $\mathbf{s}_{ij}$, surface unit normal vector $\mathbf{v}_{ij}$, and material type classification m_{ij} with i, j denoting the respective pixel column and row. From these values, as well as knowledge of the antenna position and rotation pose (which is uniform for all fragment threads), the shader can compute polar coordinates (range and azimuth angle), incidence angle, and radar cross section (RCS – explained below) in parallel.

The reference given in (Hossain et al., 2020) describes a ray tracing technique for raw SAR data generation. Although the current paper does not describe the use of ray tracing for the radar data generation, what is important is that all radar simulators that use *screen space ray tracing* should greatly benefit from the oblique camera frustum that is described in the next section.

Configuring the Conventional Symmetric Frustum

See Figure 2(a) for a visual example of the conventional symmetric frustum. The virtual camera is co-located with the virtual radar antenna, but their rotational poses are not necessarily equal. The camera's frustum must tightly enclose the terrain bounding box. To simplify the specification of the quaternion ultimately required to rotate the camera to its desired pose, engines such as Unity offer the specification of yaw, pitch, and roll angles (α, β, γ) so long as gimbal lock is not encountered. Using a left-handed orientation, where positive yaw angle moves clockwise when viewed top-down and positive pitch angle rotates downwards, they are specified as

$$\alpha = \text{atan2}\left(z_\text{far} - z_\text{c}, x_\text{far} - x_\text{c}\right), \tag{5}$$

$$\beta = \text{atan}\frac{y_\text{c} - y_\text{far}}{\sqrt{\left(x_\text{c} - x_\text{far}\right)^2 + \left(z_\text{c} - z_\text{far}\right)^2}} + \text{acos}\frac{\left(\mathbf{s}_\text{far} - \mathbf{s}_\text{c}\right) \cdot \left(\mathbf{s}_\text{close} - \mathbf{s}_\text{c}\right)}{\left\|\mathbf{s}_\text{left} - \mathbf{s}_\text{c}\right\| \left\|\mathbf{s}_\text{right} - \mathbf{s}_\text{c}\right\|}, \tag{6}$$

$$\gamma_\text{c} = 0\,. \tag{7}$$

The symmetric camera frustum has (4) specified with $D_\mathrm{r} = -D_\mathrm{l}$ and $D_\mathrm{t} = -D_\mathrm{b}$, making the 1,3 and 2, 3 matrix entries zero. Instead of directly specifying D_r, D_l, D_t, and D_b, engines such as Unity simplify the geometry for symmetric frustums by allowing specification of horizontal and vertical *field of view* (FOV) angles. The horizontal camera FOV is denoted Θ and the vertical FOV is denoted Φ and are computed as

$$\Theta = \operatorname{acos}\frac{\left(\mathbf{s}_\mathrm{left} - \mathbf{s}_\mathrm{c}\right)\cdot\left(\mathbf{s}_\mathrm{right} - \mathbf{s}_\mathrm{c}\right)}{\left\|\mathbf{s}_\mathrm{left} - \mathbf{s}_\mathrm{c}\right\|\left\|\mathbf{s}_\mathrm{right} - \mathbf{s}_\mathrm{c}\right\|}, \tag{8}$$

$$\Phi = \operatorname{acos}\frac{\left(\mathbf{s}_\mathrm{close} - \mathbf{s}_\mathrm{c}\right)\cdot\left(\mathbf{s}_\mathrm{far} - \mathbf{s}_\mathrm{c}\right)}{\left\|\mathbf{s}_\mathrm{close} - \mathbf{s}_\mathrm{c}\right\|\left\|\mathbf{s}_\mathrm{far} - \mathbf{s}_\mathrm{c}\right\|}. \tag{9}$$

The near and far plane distances are computed as

$$D_\mathrm{n} = \left\|\mathbf{s}_\mathrm{close} - \mathbf{s}_\mathrm{c}\right\|\sin\frac{\Phi}{2}, \tag{10}$$

$$D_\mathrm{f} = \left\|\mathbf{s}_\mathrm{far} - \mathbf{s}_\mathrm{c}\right\|\sin\frac{\Phi}{2}. \tag{11}$$

Radar Data Generation

The simulated complex voltage response of the scene represented by a pixel requires an approximation of that section of the scene's Radar Cross Section. Using the geometric optics flat plate approximation (McNeil et al., 2006), the RCS is

$$\sigma = \frac{\tau}{\lambda^2}A^2 \tag{12}$$

where λ is the RF carrier wavelength, A is the world-space area of the plate, and τ is the reflectivity of the material, which is a function of the wave's incident angle on the plate surface as well as the material properties. For clarity and conciseness, only two material types are used in this paper's simulated scene – the terrain material has purely diffuse scattering and the buildings are highly specular. The function τ is also termed the *scattering function* – see (Billingsley, 2002) and (Ulaby et al., 2019).

Assuming the range resolution and angle beamwidths in (1) and (2), the simulated baseband complex voltage contribution of a pixel, expressed in magnitude and phase, is approximately (Richards, 2014)

$$v = \frac{\sqrt{\sigma P}}{\sqrt{4\pi}(2r)^2}e^{\frac{-j4r}{\lambda}} \tag{13}$$

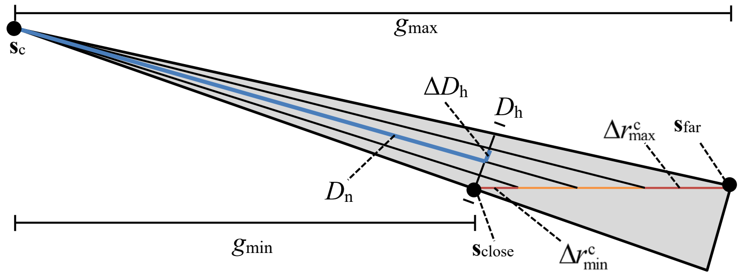
with P the transmit power, r the range to the world space plate represented by the pixel, and c the vacuum speed of propagation constant.

The three important output values per pixel are then the complex voltage v_{ij}, slant range r_{ij}, and azimuth angle ϕ_{ij}. Using these values, a parallel, many-to-one histogram binning process is implemented using a compute shader (i.e. DirectCompute, OpenGL Compute Shaders, or OpenCL), which is a regular feature of game engines such as Unity to provide post-processing of fragment shader output textures (Kaeli et al., 2015). This process creates complex-valued azimuth/range bins and sums the complex values v_{ij} depending on which range and azimuth bins they fall in (i.e. the range and azimuth values r_{ij}, ϕ_{ij} are rounded to the nearest bins). These bins are similar to what would be found in the output memory buffers of a true radar receiver's analog to digital converter. The magnitude of these bins are then trigonometrically re-mapped to a Cartesian ground plane to produce a simulated RB ground map in much the same way as performed in a real radar system (Richards, 2014). Therefore, it is the azimuth/range bins (i.e. the raw I&Q channel data) that are of primary interest since the rest of the processing mirrors the established digital signal processing of a true radar system and does not require the world scene information further.

Loss of Range Resolution with the Symmetric Frustum

Insufficient range resolution in a true radar can be thought of as a lowpass filtering process, where the reflections from closely spaced targets are blurred together into a single peak across range bins. In contrast, a loss of range resolution in the screen-space generation of raw radar data described in the previous section results from under-sampling of the scene in much the same way as an analog-to-digital converter will lose signal information if the sampling rate is below the Nyquist frequency. Because of the perspective divide operation in a conventional symmetric frustum, many radar targets in the far range of the antenna fall under the same pixel, and only one target (a building face, a point target, or a point on the ground) will be sampled by a rasterized pixel. The winning target will be that which has the lowest *z* value as is expected in the z-buffer algorithm. The use of a conventional symmetric frustum will therefore create gaps in the pure radar channel data, and ultimately the ground map image, at far range bins, as shown in Figure 1. It is recommended that N_{px} is greater than both the number of range bins and azimuth bins as a first step to mitigating this problem, though with the symmetric frustum, N_{px} may have to be impractically large to fully solve the problem.

It is important to quantify the amount of slant range difference between pixels at the far end of the range swath, $\Delta r^{\text{c}}_{\text{max}}$, as well as at the close end, $\Delta r^{\text{c}}_{\text{min}}$. See Figure 4 as a reference for the following computations. First, for this quantification formula's simplicity, the ground terrain is assumed to be a horizontal plane at sea level.



$g_{\max}$
$\mathbf{s}_{\text{c}}$
ΔD_{h}
D_{h}
$\Delta r^{\text{c}}_{\text{max}}$
$\mathbf{s}_{\text{far}}$
D_{n}
$g_{\min}$
$\mathbf{s}_{\text{close}}$
$\Delta r^{\text{c}}_{\text{min}}$

Figure 4: Diagram to aid derivation of $\Delta r^{\text{c}}_{\text{min}}$ and $\Delta r^{\text{c}}_{\text{max}}$ for the symmetric frustum. $N_{\text{px}} = 4$ for illustrative purpose. Note how the length of the ground area captured by the pixel at the top of the frustum is longer than at the bottom.

The camera's maximum range at the horizontal center of its frustum is equal to the antenna's maximum range parameter, $r^{\text{c}}_{\text{max}} = r_{\text{max}}$ and can also be defined as

$$r^{\text{c}}_{\text{max}} \equiv \left\| \mathbf{s}_{\text{c}} - \mathbf{s}_{\text{far}} \right\|, \tag{14}$$

$$r^{\text{c}}_{\text{min}} \equiv \left\| \mathbf{s}_{\text{c}} - \mathbf{s}_{\text{close}} \right\|. \tag{15}$$

Minimum and maximum ground-projected ranges are defined as

$$g_{\min} \equiv \sqrt{\left(r^{\text{c}}_{\text{min}}\right)^2 - \left(y_{\text{c}}\right)^2}, \tag{16}$$

$$g_{\max} \equiv \sqrt{\left(r^{\text{c}}_{\text{max}}\right)^2 - \left(y_{\text{c}}\right)^2}. \tag{17}$$

The near plane distance and the height of the near plane within the frustum are, respectively,

$$D_{\text{n}} = r^{\text{c}}_{\text{min}} \sin\frac{\Phi}{2}, \tag{18}$$

$$D_{\text{h}} = 2D_{\text{n}} \tan\frac{\Phi}{2}. \tag{19}$$

and each pixel divides this rectangular plate into N_{px} rows each of height $\Delta D_{\text{h}} = D_{\text{h}} / N_{\text{px}}$. Finally, the desired measures are computed as

$$\Delta r^{c}_{\text{min}} = g_{\min}\left[\sin\left(\operatorname{asin}\frac{g_{\min}}{r^{\text{c}}_{\text{min}}} + \frac{\Phi}{2} - \operatorname{atan}\frac{D_{\text{h}}/2 - \Delta D_{\text{h}}}{D_{\text{n}}}\right)\right]^{-1} - r^{\text{c}}_{\text{min}}, \tag{20}$$

$$\Delta r_{\max}^{c} = r_{\max}^{c} - g_{\min}\left[\sin\left(\operatorname{asin}\frac{g_{\min}}{r_{\min}^{c}} + \frac{\Phi}{2} + \operatorname{atan}\frac{D_{h}/2 - \Delta D_{h}}{D_{n}}\right)\right]^{-1}. \tag{21}$$

Values of (20) and (21) will be compared to those for the oblique frustum in the next section.

THE OBLIQUE CAMERA FRUSTUM SOLUTION

The properly specified oblique camera frustum significantly improves the camera sampling resolution at long ranges. See Figure 2(b). The perspective projection textures that are output from the fragment shader program under the proposed oblique frustum represent rectangular ground plane plates of nearly equal length and width (terrain elevation features slightly perturb the uniformity of the length and width of the plates). This way, the radar simulation maintains consistent ground range resolution specified by the pixel resolution of the perspective projection texture. The oblique camera frustum contains a linear shearing operation by modifying the 1,3 and 2,3 entries of (4) to be nonzero.

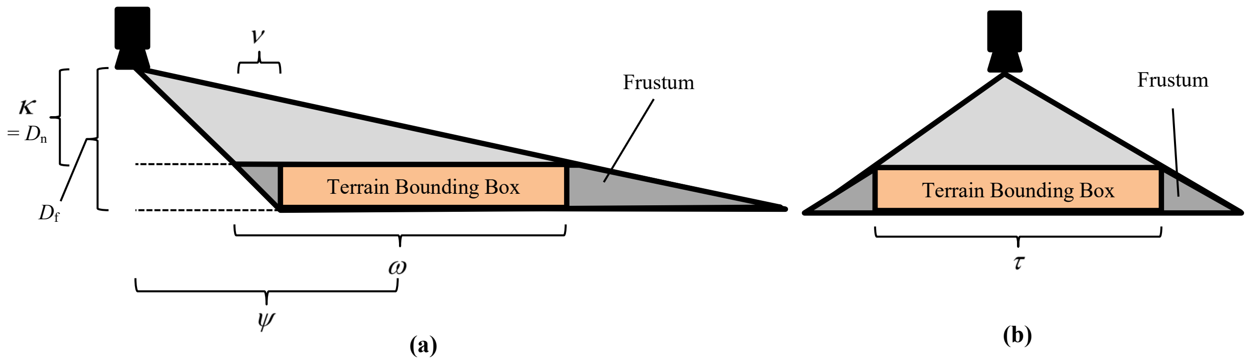


Figure 5: Visual description of the parameters required to set up the oblique frustum (compatible with Unity). (a) - Side view. (b) Head-on view as seen from behind the radar antenna.

The camera's yaw angle is the same as in (5) but the pitch angle is always $90°$. Since the camera is pointing downwards, the camera's z axis is inversely aligned with the world y axis. Simulation engines such as Unity can be set up to have an oblique camera frustum by conveniently specifying a set of distinct parameters called *physical parameters* which are familiar in photography.

Using Figures 3 and 5 as references, the new required camera parameters are: κ (focal length), τ (camera sensor size in camera dimension x), ω (camera sensor size in camera dimension y, and ψ (the lens shift in camera dimension y). The camera sensor is elongated from the size of the bounding box in the camera y dimension by ν. There is no elongation or lens shift in the camera x dimension. The following geometric relationships are formed:

$$\kappa = y_{c} - y_{tb} - W_{y} \tag{22}$$

$$D_{n} = \kappa \tag{23}$$

$$D_{f} = D_{n} + W_{y} \tag{24}$$

$$\tau = W_{x} \tag{25}$$

$$\nu = -z_{c}\frac{W_{y}}{y_{c}} \tag{26}$$

$$\omega = (w_{z} + \nu) \tag{27}$$

$$\psi = \frac{1}{2}(W_{z} + \nu) + z_{c} - \nu \tag{28}$$

For simplicity of derivation and implementation, the near clip plane distance in (23) is the same as the focal length distance. The combination of the focal plane distance and the sensor size parameters determine the angle field of view of the camera in both viewing dimensions.

It is important to note that some azimuth sampling resolution is lost at close range when using the oblique frustum, much in the same way as range information is lost at far range when using the symmetric frustum, but the effect is more easily mitigated by sampling the perspective projection at a higher pixel resolution than the range sampling resolution is. Because of the Cartesian ground plane projection of the final image, this tradeoff is not noticeable. If polar space raw radar channel data is important for purposes other than ground map simulation, the sampling stage can be designed to scan the camera over the scene with higher pixel resolution at closer ranges.

The slant range difference between pixels, $\Delta r_{\max}^{c}$ and $\Delta r_{\min}^{c}$, are more easily computed for the horizontally oriented oblique frustum:

$$\Delta r_{\min}^{c} = \sqrt{\left[g_{\min} + \frac{\left(g_{\max} - g_{\min}\right)}{N_{\mathrm{px}}}\right]^2 + y_{\mathrm{c}}^2} - r_{\min}^{c} \tag{29}$$

$$\Delta r_{\max}^{c} = r_{\min}^{c} - \sqrt{\left[g_{\max} - \frac{\left(g_{\max} - g_{\min}\right)}{N_{\mathrm{px}}}\right]^2 + y_{\mathrm{c}}^2} \tag{30}$$

Figure 6 plots $\Delta r_{\min}^{c}$ and $\Delta r_{\max}^{c}$ for the symmetric and oblique frustums as a comparison, with the antenna's $r_{\max}$ varied from 10 km to 100 km while the antenna is fixed at an altitude of one kilometer. The number of pixel rows N_{px} used is 2048 and it is assumed, for simplicity, that $r_{\min}^{c} = r_{\max}/10$.

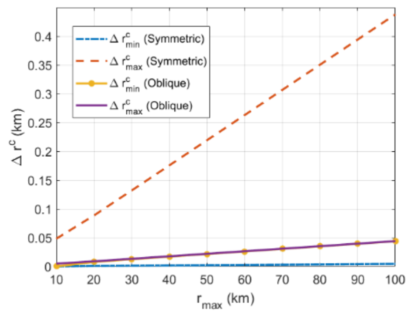


Figure 6: Comparison of the range difference represented by pixels at the far end of the range swath and the close end of the range swath for the conventional symmetric frustum and the proposed oblique frustum.

Figure 6 clearly shows the poor performance of the symmetric frustum, since its $\Delta r_{\max}^{c}$ can be as poor as half of a kilometer at the far end of the range swath, while at the close range, its $\Delta r_{\max}^{c}$ remains extremely low. This disparity in sampling resolution is undesirable. The symmetric frustum, however, shows nearly equal sampling resolution throughout the range swath no matter what the size and extend of the swath is and this is highly desirable.

SIMULATION RESULTS

A scene in Texas, USA is modeled utilizing publicly available elevation data from the U.S. Geological Survey (USGS) National Map. Building meshes are added by randomly placing rectangular prism objects in areas identified as containing buildings. The antenna has an altitude of $y_{\mathrm{c}} = 1$ km, the range extent is $r_{\max} = 37$ km, and the angle sector is $\theta_{\mathrm{sec}} = 120°$. The number of pixel rows and columns in perspective projection sampling data are both $N_{\mathrm{px}} = 2048$ and there are 1024 range bins and 1024 azimuth bins. Simulated real beam ground maps are shown in Figure 1. Note that in these simulations, and similar to true radar imaging, the received power is normalized to remove its dependence on range, making the brightness of the image consistent throughout the range swath. It can be seen how use of the oblique camera frustum maintains more information in the final ground map image due to the consistent range sampling resolution. Furthermore, it can be seen that not only is range information missing in the symmetric frustum version, but many targets (the building reflectors) do not get sampled in the simulation as can be seen by the missing bright white spots in Figure 1a and 1c, while many more targets appear when using the oblique frustum as seen in 1b and 1d.

CONCLUSIONS

This paper has described how to utilize the GPU rasterization pipeline to efficiently solve the visibility problem for simulated radar data generation while not compromising the achievable range resolution provided by the virtual camera's perspective projection output data. The use of the rasterization pipeline to generate raw radar channel data was described, which produces geometrically correct, realistic radar images that naturally include shadow, overlay, and speckle. Methods to configure the conventional symmetric frustum and the proposed horizontally aligned oblique frustum were presented and formulas were derived to numerically show the advantage of the oblique frustum. A simulated scenario was described and final output images were presented that visually compare the drawback of using a symmetric frustum, while it was shown that the oblique frustum solves the same visibility problem but provides consistent range sampling throughout the range swath. These techniques greatly advance the technology for providing high-fidelity, realistic, real-time radar image generation, as well as raw radar data generation for data analysis applications.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the support of USAF personnel at AFWERX, the Air Force Life Cycle Management Center, Simulator Program office (AFLCMC/WNS) and the 7th Bomb Wing. ISL responded to and won a Phase I and subsequent Phase II effort under the AFWERX Small Business Innovation Research (SBIR) Special Topic AF192-001 – "Open Call for Innovative Defense-Related Dual-Purpose Technologies/Solutions with a Clear Air Force Stakeholder Need," proposing the Radar/Radio Frequency (RF)/Electronic Warfare (EW) Sidecar.

REFERENCES

Balster, E. J., Scarpino, F. A., Kordik, A. M., & Hill, K. L. (2017). Synthetic aperture radar imaging simulator for pulse envelope evaluation. *Journal of Applied Remote Sensing*, *11*(4).

Balz, T., Hammer, H., & Auer, S. (2015). Potentials and limitations of SAR image simulators – A comparative study of three simulation approaches. *ISPRS Journal of Photogrammetry and Remote Sensing*, *101*, 102-109.

Billingsley, J. B. (2002). *Low-angle Radar Land Clutter: Measurements and Empirical Models*. IET.

Hossain, S., Willis, A. R., & Godwin, J. (2020). Hardware-accelerated SAR simulation with NVIDIA-RTX technology. Algorithms for Synthetic Aperture Radar Imagery XXVII,

Kaeli, D. R., Mistry, P., Schaa, D., & Zhang, D. P. (2015). *Heterogeneous Computing with OpenCL 2.0* (3 ed.). Morgan Kaufmann.

Lengyel, E. (2012). *Mathematics for 3D Game Programming and Computer Graphics* (Third ed.). Course Technologt PTR.

Lin, Y., Xue, S., & Chen, D. F. (2016). Acceleration for the real-time radar echoes simulation of space multi-target using GPU. 2nd IEEE International Conference on Computer and Communications (ICCC), Chengdu, China.

Martelli, M., Enderli, C., Gac, N., Vermesse, A., & Mérigot, A. (2019). GPU Acceleration : OpenACC for Radar Processing Simulation. International Radar Conference, Toulon, Fance.

McNeil, S. C., Bergin, J. S., & Techau, P. (2006). Modeling the impact of discrete clutter on airborne adaptive radar systems. 2006 IEEE Conference on Radar,

Peinecke, N., Döhler, H.-U., & Korn, B. (2008). Simulation of Imaging Radar Using Graphics Hardware Acceleration. Proceedings of SPIE - The International Society for Optical Engineering,

Richards, M. A. (2014). *Fundamentals of Radar Signal Processing*. McGraw-Hill Education.

Schöffmann, C., Ubezio, B., Böhm, C., Mühlbacher-Karrer, S., & Zangl, H. (2021). Virtual Radar: Real-Time Millimeter-Wave Radar Sensor Simulation for Perception-Driven Robotics. *IEEE Robotics and Automation Letters*, *6*(3), 4704-4711.

Ulaby, F. T., Dobson, M. C., & Álvarez-Pérez, J. L. (2019). *Handbook of Radar Scattering Statistics for Terrain*. Artech House.

Zhang, L., Yu, Y., Xu, T., & Zhang, J. (2013). Simulation of Airborne Radar Real Beam Ground Map Based on Digital Terrain. 2013 International Conference on Computational and Information Sciences, Shiyang, China.