Samurai Shading in Ghost of Tsushima
Supplemental Material

Jasmin Patry

1 ANISOTROPIC ASPERITY SCATTERING BRDF IMPLEMENTATION DETAILS

This section contains implementation details for the anisotropic asperity scattering BRDF. Please note that we only deal with the version of the BRDF that uses the SGGX phase function here.

1.1 Approximation of P_s

Our implementation uses several approximations for integrals with no known closed-form solution, found using curve fitting. The first of these is for P_s, the fraction of light scattered by the scattering layer toward the base layer.

Our approximate function $\widetilde{P}_s$ is given by

$$P_s(\hat{\mathbf{u}}) \approx \widetilde{P}_s(\hat{\mathbf{u}}) = c_0 + c_1 \hat{\mathbf{u}} \cdot \hat{\mathbf{n}} + c_2 \hat{\mathbf{u}} \cdot \hat{\mathbf{t}} + c_3 (\hat{\mathbf{u}} \cdot \hat{\mathbf{n}})^2 + c_4 (\hat{\mathbf{u}} \cdot \hat{\mathbf{n}})(\hat{\mathbf{u}} \cdot \hat{\mathbf{t}}) + c_5 (\hat{\mathbf{u}} \cdot \hat{\mathbf{t}})^2, \tag{1}$$

where the c_i terms are constants found by curve fitting. These constants also depend on α and θ_f, which we restrict to be non-spatially varying. The function therefore depends on four variables: $\theta_{\hat{\mathbf{u}}}$, the light polar angle; $\phi_{\hat{\mathbf{u}}}$, the azimuthal angle of the light as measured from the fiber direction $\hat{\mathbf{t}}$; α, the projected area of microflakes in the fiber direction $\hat{\mathbf{t}}$; and θ_f, the fiber polar angle.

Our approach to fitting this function involved finding c_i values, via curve fitting, for each point in a 65×65 grid of α and θ_f values. These could be used as a LUT, but we opted for a more compact approach: we observed from these samples that the c_i are smooth functions of α and θ_f, so we fit curves to them as well. (By using this approach we turned a potentially intractable 4D curve fitting problem into a set of much smaller 2D curve fitting problems.) To compute each c_i we used

$$c_i = s_i + \beta \left(\frac{t_i + \mathbf{p}_i^T \mathbf{x}}{1 + \mathbf{q}_i^T \mathbf{x}} \right), \tag{2}$$

where

$$\begin{aligned}
\mathbf{x} &= \begin{bmatrix} \beta & \gamma & \delta & \beta^2 & \gamma^2 & \beta\gamma & \beta\delta & \beta^3 & \gamma^3 & \delta^3 & \beta^2\gamma & \beta^2\delta & \beta\gamma^2 \end{bmatrix}^T, \\
\beta &= 1 - \alpha, \\
\gamma &= \cos\theta_f, \\
\delta &= \sin\theta_f, \\
s_i &= \begin{cases} \frac{1}{2} & i = 0, \\ 0 & \text{otherwise}, \end{cases}
\end{aligned}$$

and the components of the thirteen-element vectors $\mathbf{p}_i$ and $\mathbf{q}_i$, along with t_i, are constant coefficients found by curve fitting for each $i \in [0, 5]$. (Note that in our implementation, Equation 2 is evaluated at shader compilation time.) We provide values for these constants, together with code for evaluation of Equation 2, in Appendix A; plots of $P_s(\hat{\mathbf{u}})$ and $\widetilde{P}_s(\hat{\mathbf{u}})$ are provided in Appendix B.

1.2 Ambient Lighting Approximation

The next integral that we approximate is used by the ambient lighting model. It is the average value over the hemisphere of the BRDF of the scattering layer — excluding (i.e., divided by) the fiber albedo and phase function — which we'll call G:

$$\begin{aligned} G(d, \hat{\mathbf{v}}) &= \frac{1}{2\pi} \int_\Omega \frac{f(\hat{\mathbf{u}}, \hat{\mathbf{v}})}{\mathbf{c}_f p(\hat{\mathbf{u}}, \hat{\mathbf{v}})} (\hat{\mathbf{u}} \cdot \hat{\mathbf{n}}) \, \mathrm{d}\omega_{\hat{\mathbf{u}}} \\ &= \frac{1}{2\pi} \int_\Omega \frac{1 - \exp\left(-d \frac{(\hat{\mathbf{u}}+\hat{\mathbf{v}})\cdot\hat{\mathbf{n}}}{(\hat{\mathbf{u}}\cdot\hat{\mathbf{v}})(\hat{\mathbf{v}}\cdot\hat{\mathbf{n}})}\right)}{(\hat{\mathbf{u}} + \hat{\mathbf{v}}) \cdot \hat{\mathbf{n}}} (\hat{\mathbf{u}} \cdot \hat{\mathbf{n}}) \, \mathrm{d}\omega_{\hat{\mathbf{u}}} \\ &= \frac{1}{2\pi} \int_\Omega g(\hat{\mathbf{u}}, \hat{\mathbf{v}}) \, \mathrm{d}\omega_{\hat{\mathbf{u}}} \\ &\approx \widetilde{G}(d, \hat{\mathbf{v}}) = \frac{d(1 + c_0 \hat{\mathbf{v}} \cdot \hat{\mathbf{n}} + c_1 (\hat{\mathbf{v}} \cdot \hat{\mathbf{n}})^2)}{d + c_2 \hat{\mathbf{v}} \cdot \hat{\mathbf{n}} + c_3 (\hat{\mathbf{v}} \cdot \hat{\mathbf{n}})^3}, \end{aligned} \tag{3}$$

where d is the density of the scattering layer, and the c_i terms are constants found by curve fitting. We provide values for these constants, together with code for evaluation of Equation 3, in Appendix C; plots of $G(d, \hat{\mathbf{v}})$ and $\widetilde{G}(d, \hat{\mathbf{v}})$ are provided in Appendix D.

1.3 Ambient Shadowing Approximation of Average P_p

Next, we approximate the cosine-weighted average of P_p (the probability of a ray penetrating to the base layer without being scattered by the scattering layer) over the hemisphere, which we'll call Q:

$$\begin{aligned} Q(d) &= \frac{1}{\pi} \int_\Omega P_p(\hat{\mathbf{u}}) (\hat{\mathbf{u}} \cdot \hat{\mathbf{n}}) \, \mathrm{d}\omega_{\hat{\mathbf{u}}} \\ &\approx \widetilde{Q}(d) = 1 + c_0 d + c_1 d^2, \end{aligned} \tag{4}$$

where d is the density of the scattering layer, and c_0 and c_1 are constants found by curve fitting. We provide values for these constants, together with code for evaluation of Equation 4, in Appendix E; a plot of $Q(d)$ and $\widetilde{Q}(d)$ are provided in Appendix F.

1.4 Ambient Shadowing Approximation of Average P_s

Finally, we approximate the cosine-weighted average of P_s over the hemisphere, which we'll call R:

$$\begin{aligned} R(\theta_f, \alpha) &= \frac{1}{\pi} \int_\Omega P_s(\hat{\mathbf{u}}) (\hat{\mathbf{u}} \cdot \hat{\mathbf{n}}) \, \mathrm{d}\omega_{\hat{\mathbf{u}}} \\ &\approx \widetilde{R}(\theta_f, \alpha) = \frac{1}{2} + \beta \left(\frac{t + \mathbf{p}^T \mathbf{x}}{1 + \mathbf{q}^T \mathbf{y}} \right), \end{aligned} \tag{5}$$

where

$$\begin{aligned} \mathbf{x} &= \begin{bmatrix} \beta & \gamma & \delta & \beta^2 & \gamma^2 & \beta\gamma & \beta\delta & \beta^3 & \gamma^3 & \delta^3 & \beta^2\gamma & \beta^2\delta & \beta\gamma^2 \end{bmatrix}^T, \\ \mathbf{y} &= \begin{bmatrix} \beta & \gamma & \delta \end{bmatrix}^T, \\ \beta &= 1 - \alpha, \\ \gamma &= \cos\theta_f, \\ \delta &= \sin\theta_f, \end{aligned}$$

and the components of the thirteen-element vector $\mathbf{p}$ and three-element vector $\mathbf{q}$, along with t, are constant coefficients found by curve fitting. (Note that in our implementation, Equation 5 is evaluated at shader compilation time.) We provide values for these constants, together with code for evaluation of Equation 5, in Appendix G; plots of R and $\widetilde{R}$ are provided in Appendix H.

REFERENCES

[Dup+13] J. Dupuy, E. Heitz, J.-C. Iehl, P. Poulin, F. Neyret, and V. Ostromoukhov. "Linear Efficient Antialiased Displacement and Reflectance Mapping". In: *ACM Trans. Graph.* 32.6 (Nov. 2013). ISSN: 0730-0301. DOI: 10.1145/2508363.2508422. URL: https://hal.inria.fr/hal-00858220v1/.

[Hei+15] E. Heitz, J. Dupuy, C. Crassin, and C. Dachsbacher. "The SGGX Microflake Distribution". In: *ACM Trans. Graph.* 34.4 (July 2015). ISSN: 0730-0301. DOI: 10.1145/2766988. URL: https://eheitzresearch.wordpress.com/research/.

[KK89] J. T. Kajiya and T. L. Kay. "Rendering Fur with Three Dimensional Textures". In: *SIGGRAPH Comput. Graph.* 23.3 (July 1989), pp. 271–280. ISSN: 0097-8930. DOI: 10.1145/74334.74361. URL: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.127.5564.

[KP03] J. Koenderink and S. Pont. "The secret of velvety skin". English. In: *Machine Vision and Applications* 14.4 (2003), pp. 260–268. ISSN: 0932-8092. DOI: 10.1007/s00138-002-0089-7. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.105.1847&rep=rep1&type=pdf.

[McA+12] S. McAuley, S. Hill, N. Hoffman, Y. Gotanda, B. Smits, B. Burley, and A. Martinez. "Practical Physically-Based Shading in Film and Game Production". In: *ACM SIGGRAPH 2012 Courses*. SIGGRAPH '12. Los Angeles, California: Association for Computing Machinery, 2012. ISBN: 9781450316781. DOI: 10.1145/2343483.2343493. URL: https://blog.selfshadow.com/publications/s2012-shading-course/.

[McA13] S. McAuley. *Extension to Energy-Conserving Wrapped Diffuse*. http://blog.stevemcauley.com/2013/01/30/extension-to-energy-conserving-wrapped-diffuse/. Jan. 2013.

[OB10] M. Olano and D. Baker. "LEAN Mapping". In: *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '10. Washington, D.C.: Association for Computing Machinery, 2010, pp. 181–188. ISBN: 9781605589398. DOI: 10.1145/1730804.1730834. URL: https://www.csee.umbc.edu/~olano/papers/lean/.

[PB11] E. Penner and G. Borshukov. "Pre-Integrated Skin Shading". In: *GPU Pro 2*. Ed. by W. Engel. 1st. New York: A K Peters/CRC Press, 2011. Chap. 1. ISBN: 9780429108488.

[PH10] M. Pharr and G. Humphreys. "Physically Based Rendering, Second Edition: From Theory To Implementation". In: 2nd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010, pp. 583–587. ISBN: 0123750792, 9780123750792.

[RH01a] R. Ramamoorthi and P. Hanrahan. "An Efficient Representation for Irradiance Environment Maps". In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. New York, NY, USA: ACM, 2001, pp. 497–500. ISBN: 1-58113-374-X. DOI: 10.1145/383259.383317. URL: https://cseweb.ucsd.edu/~ravir/papers/envmap/.

[RH01b] R. Ramamoorthi and P. Hanrahan. "On the relationship between Radiance and Irradiance: Determining the illumination from images of a convex Lambertian object". In: *Journal of the Optical Society of America* 18.10 (Oct. 2001), pp. 2448–2459. URL: https://cseweb.ucsd.edu/~ravir/papers/invlamb/.

[Rus04] S. Rusinkiewicz. "Estimating Curvatures and Their Derivatives on Triangle Meshes". In: *Symposium on 3D Data Processing, Visualization, and Transmission*. Sept. 2004. URL: https://gfx.cs.princeton.edu/pubs/Rusinkiewicz_2004_ECA/index.php.

[RV11] P. Ramachandran and G. Varoquaux. "Mayavi: 3D Visualization of Scientific Data". In: *Computing in Science & Engineering* 13.2 (2011), pp. 40–51. ISSN: 1521-9615.

Appendices

A $\widetilde{P}_S$ CONSTANTS AND EVALUATION CODE

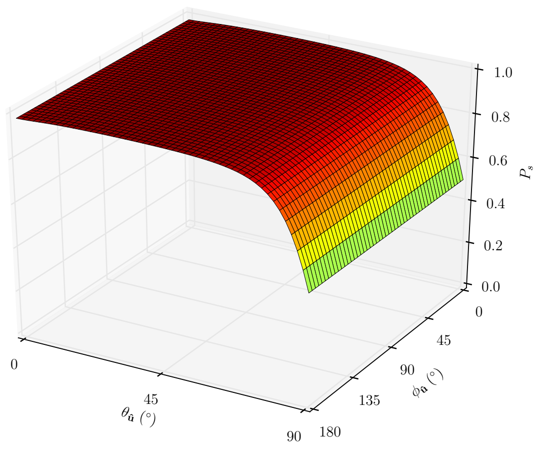
```cpp
static const float s_PsCoeffs[6][27] =
{
    {
        0.13756974f, 0.089967924f, -0.043263807f, 0.012549466f, -0.039043481f, -0.18738464f,
        0.019518992f, -0.14037448f, -0.018000072f, 0.080983287f, -0.12573283f, 0.030190086f,
        0.082546164f, -0.067284671f, -1.4158549f, -0.8067329f, -0.18696706f, 0.85672806f,
        0.73143116f, 0.15639405f, -0.3532818f, -0.57225366f, -0.12095449f, 0.036308587f,
        0.72092589f, 0.60578342f, -0.58250484f,
    },
    {
        -2.7184817f, -5.8696774f, 8.0163004f, -3.1600727f, 3.8612209f, -5.1905628f,
        -5.0022991f, 10.047277f, 0.18762356f, 0.37025318f, 2.4688789f, -2.4440128f,
        -4.838545f, 8.5374892f, -8.0595452f, -17.180627f, 1.0716076f, 4.4852769f,
        29.035032f, 9.6089763f, -0.85605057f, -2.6545399f, -7.9458921f, 4.995542f,
        5.0528634f, 0.037475838f, -13.311191f,
    },
    {
        -0.78639883f, 1.5746798f, 2.48243f, 1.3541003f, -0.95093934f, -4.1094587f,
        -2.463887f, -1.4678292f, 0.051742457f, 2.3265071f, -0.58747343f, 0.95003155f,
        0.80560378f, 0.92021804f, -0.62641789f, -4.2275626f, 1.2757363f, -0.20707868f,
        3.8802392f, 2.9039937f, -2.079426f, -0.067985356f, -0.73187572f, 0.13138015f,
        0.3012767f, 1.0202816f, -2.2280816f,
    },
    {
        -0.24041982f, -0.13259461f, -0.38462128f, 0.026466062f, 0.19445785f, 1.0051305f,
        0.48317105f, 0.10568307f, 0.012401867f, -0.44645249f, 0.22808668f, -0.23192799f,
        -0.19285076f, -0.2408241f, -1.5432125f, -1.3051729f, -0.056543476f, 0.90136729f,
        0.4311222f, 1.6008327f, 0.11762443f, -0.17825185f, 0.016131527f, -0.11530679f,
        -0.44470983f, -0.074847149f, -0.47874924f,
    },
    {
        0.24779292f, 1.7000882f, -2.1202227f, 1.0560525f, -1.8178813f, 1.2658071f,
        2.3142857f, -3.0859394f, 0.29918169f, 0.30209918f, -0.33373112f, 0.11488872f,
        1.8913286f, -2.3312817f, -10.140274f, -14.579435f, 1.7571963f, 6.8424411f,
        18.456914f, 18.485136f, 1.7617207f, -0.36251123f, -2.8078648f, 2.7987085f,
        -3.2621689f, -4.0289481f, -13.375577f,
    },
    {
        -1.7111756f, 1.1697807f, -1.3136888f, 0.29216331f, -2.4200667f, 5.5264719f,
        4.008041f, -1.9572698f, -0.0043264307f, -2.5938981f, 1.3983587f, 0.29401827f,
        2.7673795f, -4.331847f, 19.422409f, -3.7802382f, 5.591913f, -17.588813f,
        6.7184933f, -19.919567f, -18.57141f, 0.13662077f, -3.5379282f, -1.3720354f,
        15.545388f, 14.697374f, 2.0091252f,
    },
};
```

Listing 1: C++ table of coefficients for Equation 2.

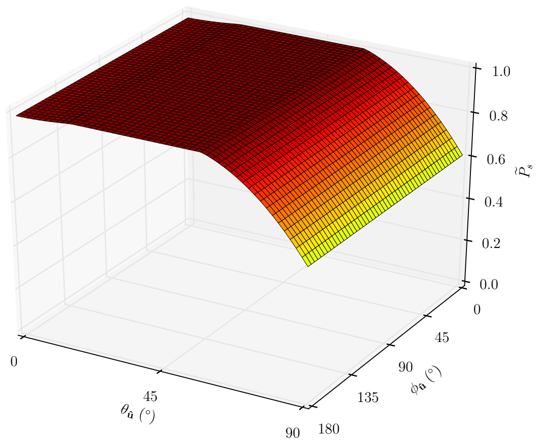
```cpp
void CalcFuzzPsCoeffs(float alpha, float thetaF, float (& c)[6])
{
    // Clamp to the domain used for curve fitting. thetaF values less than 0.1
    //  radians (~5.7 degrees) were not used because the function becomes weakly
    //  dependent on phi_u in that region, which causes the curve fitting
    //  parameters to diverge.

    static const float s_thetaFMin = 0.1f;
    thetaF = max(s_thetaFMin, min(pi * 0.5f, thetaF));

    static const float s_alphaMin = 0.15f;
    alpha = max(s_alphaMin, min(1.0f, alpha));

    float sinThetaF = sinf(thetaF);
    float cosThetaF = cosf(thetaF);

    // beta, gamma, delta

    float b = 1.0f - alpha;
    float g = cosThetaF;
    float d = sinThetaF;

    float b2 = b * b;
    float b3 = b2 * b;

    float g2 = g * g;
    float g3 = g2 * g;

    float d2 = d * d;
    float d3 = d2 * d;

    for (int i = 0; i < 6; ++i)
    {
        float s = (i == 0) ? 0.5f : 0.0f;
        const float * p = &s_PsCoeffs[i][0];
        float num = p[0] + p[1] * b + p[2] * g + p[3] * d + p[4] * b2 + p[5] * g2 +
                        p[6] * b * g + p[7] * b * d + p[8] * b3 + p[9] * g3 +
                        p[10] * d3 + p[11] * b2 * g + p[12] * b2 * d + p[13] * b * g2;
        float den = 1.0f + p[14] * b + p[15] * g + p[16] * d + p[17] * b2 + p[18] * g2 +
                        p[19] * b * g + p[20] * b * d + p[21] * b3 + p[22] * g3 +
                        p[23] * d3 + p[24] * b2 * g + p[25] * b2 * d + p[26] * b * g2;
        c[i] = s + b * num / den;
    }
}
```

Listing 2: C++ implementation of Equation 2.

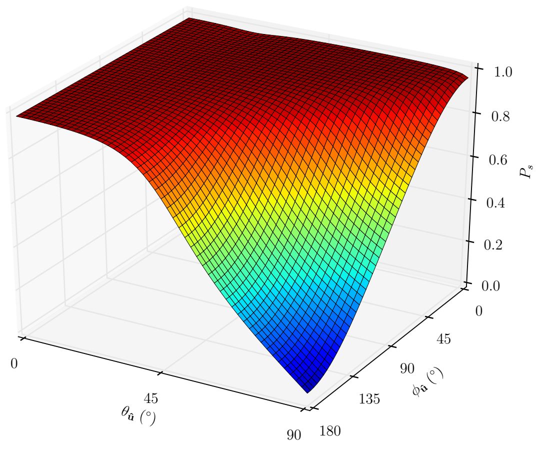
B PLOTS OF P_S AND $\widetilde{P}_S$



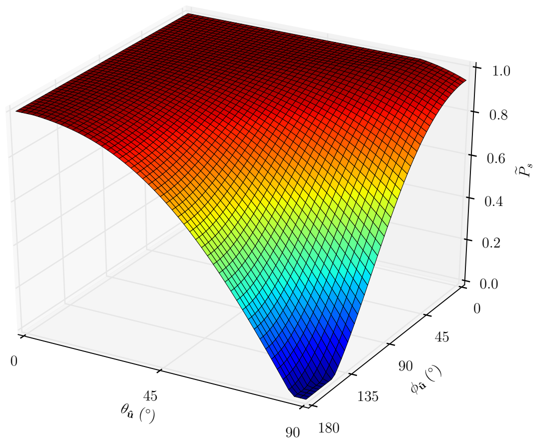
(a) $P_s(\hat{\mathbf{u}})$ with $\theta_f = 0°$.



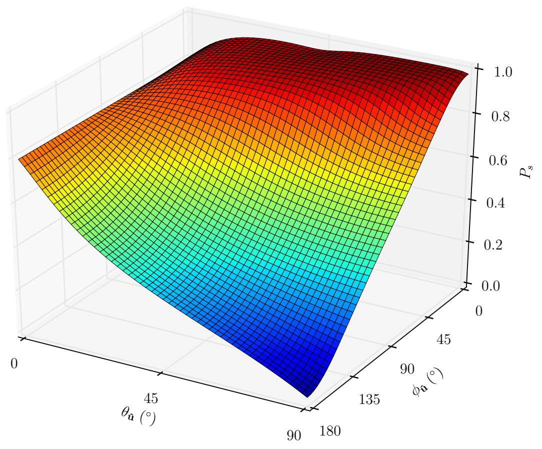
(b) $\widetilde{P}_s(\hat{\mathbf{u}})$ with $\theta_f = 0°$.



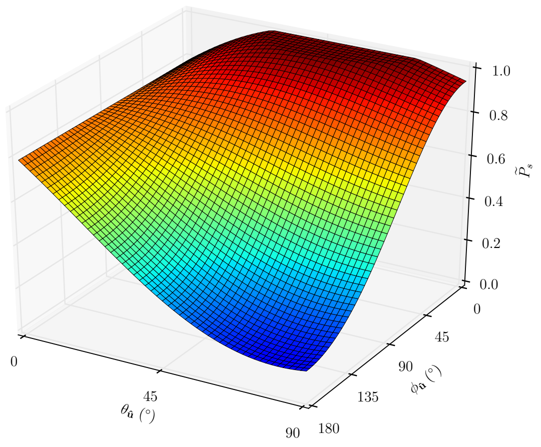
(c) $P_s(\hat{\mathbf{u}})$ with $\theta_f = 23°$.



(d) $\widetilde{P}_s(\hat{\mathbf{u}})$ with $\theta_f = 23°$.

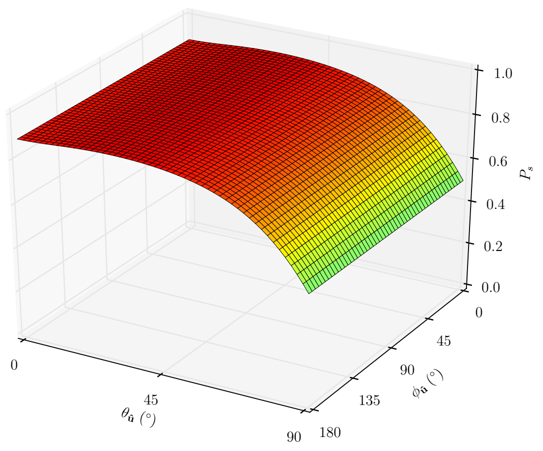


(e) $P_s(\hat{\mathbf{u}})$ with $\theta_f = 45°$.

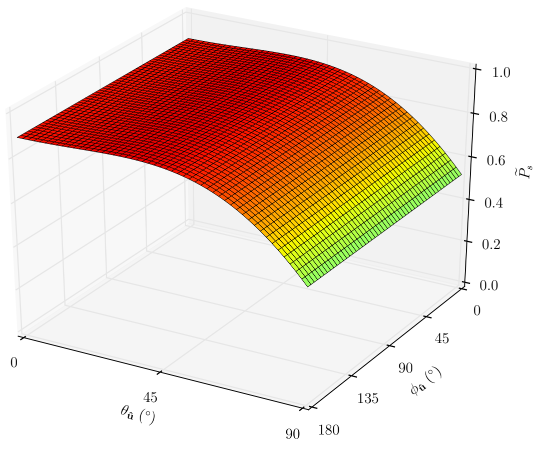


(f) $\widetilde{P}_s(\hat{\mathbf{u}})$ with $\theta_f = 45°$.

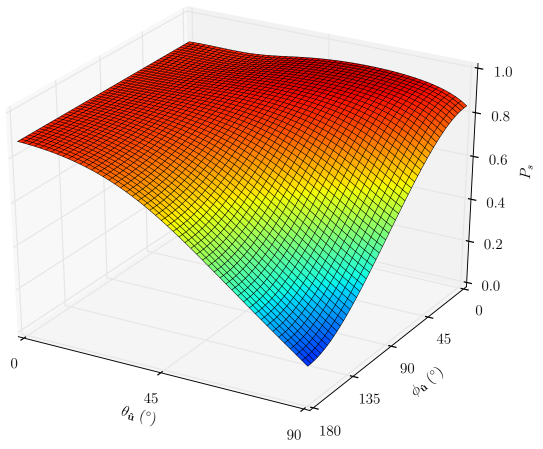
Figure 1: Plots of $P_s(\hat{\mathbf{u}})$ and $\widetilde{P}_s(\hat{\mathbf{u}})$ with $\alpha = 0.15$.



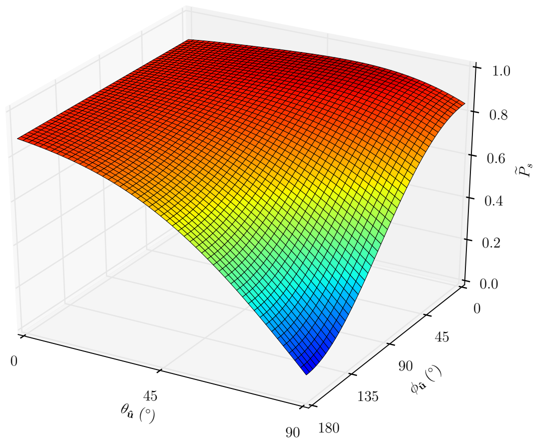
(a) $P_s(\hat{\mathbf{u}})$ with $\theta_f = 0°$.



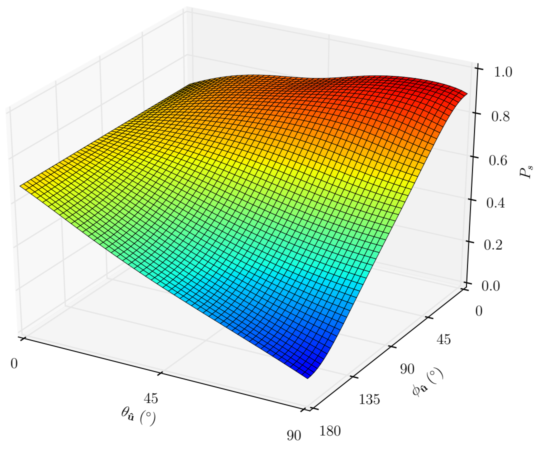
(b) $\widetilde{P}_s(\hat{\mathbf{u}})$ with $\theta_f = 0°$.



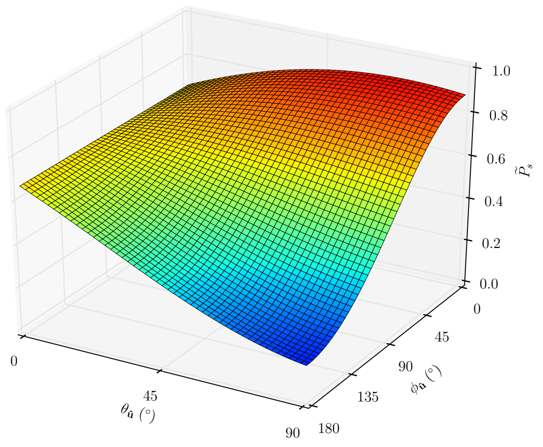
(c) $P_s(\hat{\mathbf{u}})$ with $\theta_f = 23°$.



(d) $\widetilde{P}_s(\hat{\mathbf{u}})$ with $\theta_f = 23°$.

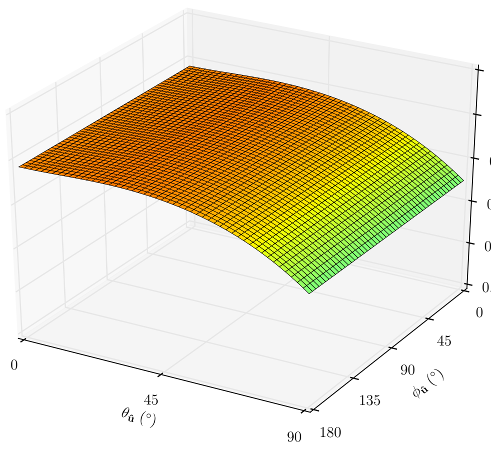


(e) $P_s(\hat{\mathbf{u}})$ with $\theta_f = 45°$.

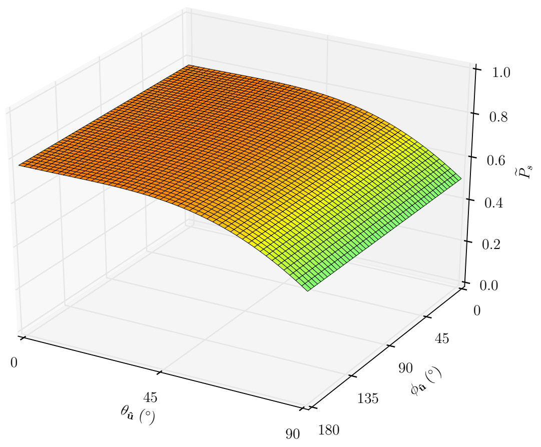


(f) $\widetilde{P}_s(\hat{\mathbf{u}})$ with $\theta_f = 45°$.

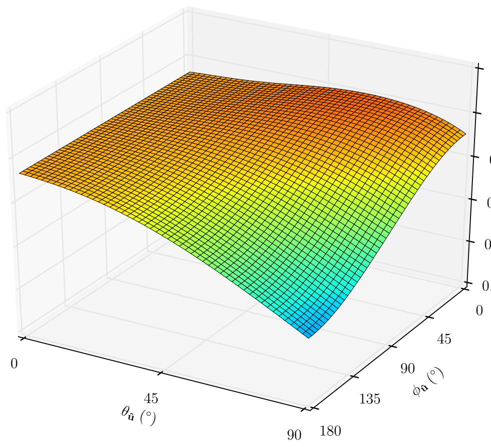
Figure 2: Plots of $P_s(\hat{\mathbf{u}})$ and $\widetilde{P}_s(\hat{\mathbf{u}})$ with $\alpha = 0.35$.



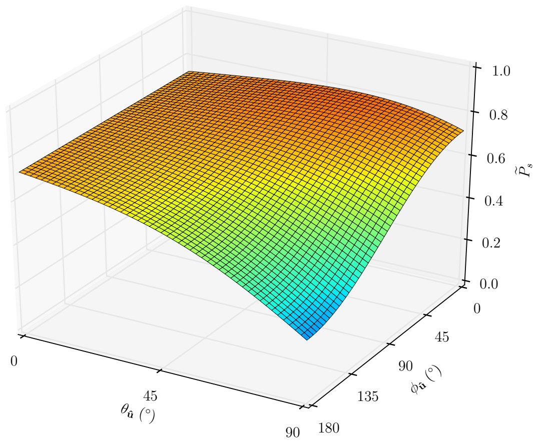
(a) $P_s(\hat{\mathbf{u}})$ with $\theta_f = 0°$.



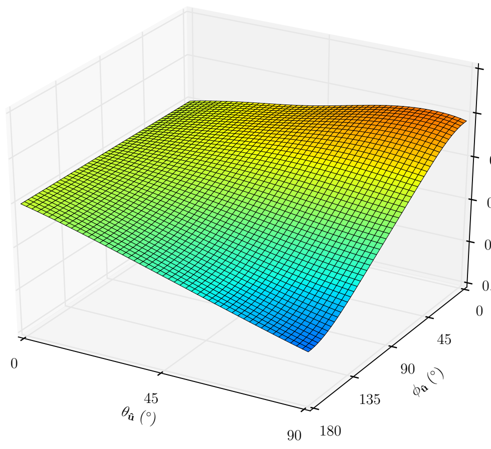
(b) $\widetilde{P}_s(\hat{\mathbf{u}})$ with $\theta_f = 0°$.



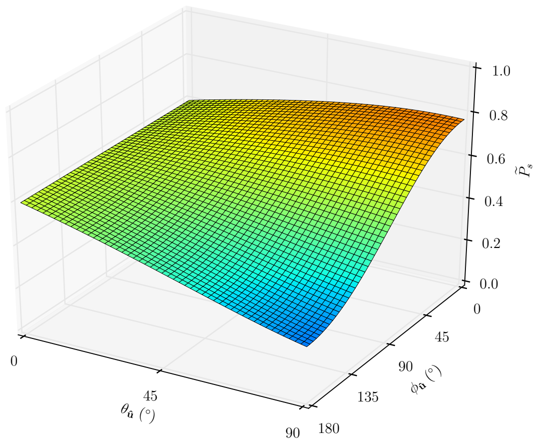
(c) $P_s(\hat{\mathbf{u}})$ with $\theta_f = 23°$.



(d) $\widetilde{P}_s(\hat{\mathbf{u}})$ with $\theta_f = 23°$.

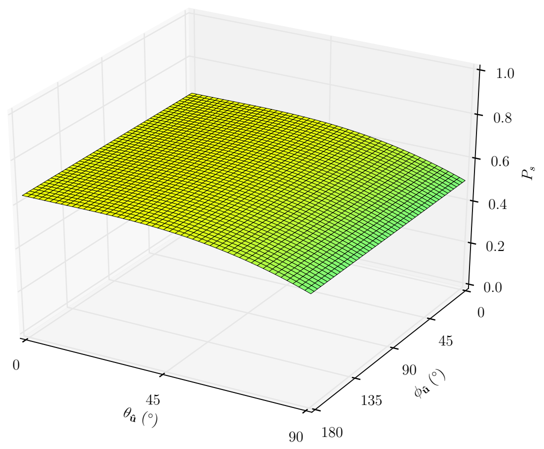


(e) $P_s(\hat{\mathbf{u}})$ with $\theta_f = 45°$.

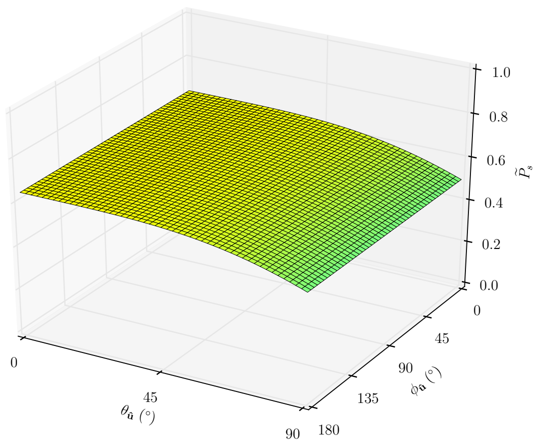


(f) $\widetilde{P}_s(\hat{\mathbf{u}})$ with $\theta_f = 45°$.

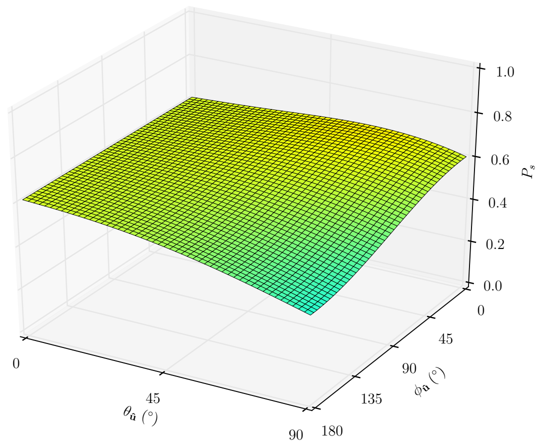
Figure 3: Plots of $P_s(\hat{\mathbf{u}})$ and $\widetilde{P}_s(\hat{\mathbf{u}})$ with $\alpha = 0.55$.



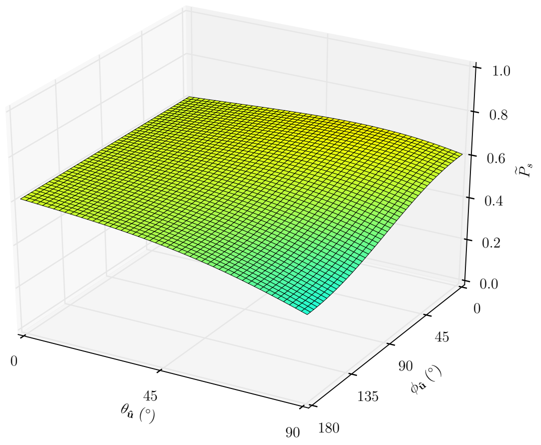
(a) $P_s(\hat{\mathbf{u}})$ with $\theta_f = 0°$.



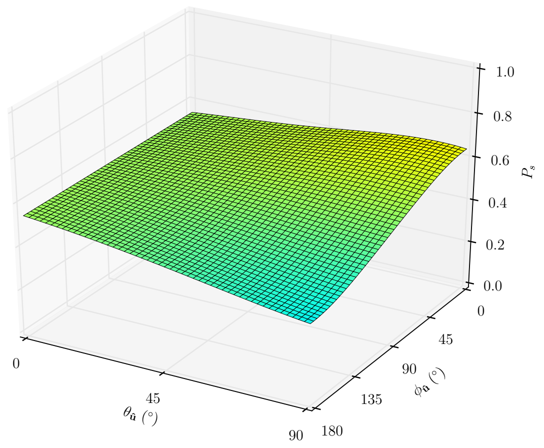
(b) $\widetilde{P}_s(\hat{\mathbf{u}})$ with $\theta_f = 0°$.



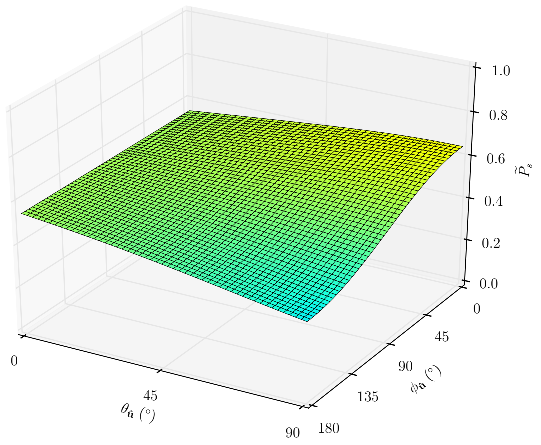
(c) $P_s(\hat{\mathbf{u}})$ with $\theta_f = 23°$.



(d) $\widetilde{P}_s(\hat{\mathbf{u}})$ with $\theta_f = 23°$.



(e) $P_s(\hat{\mathbf{u}})$ with $\theta_f = 45°$.



(f) $\widetilde{P}_s(\hat{\mathbf{u}})$ with $\theta_f = 45°$.

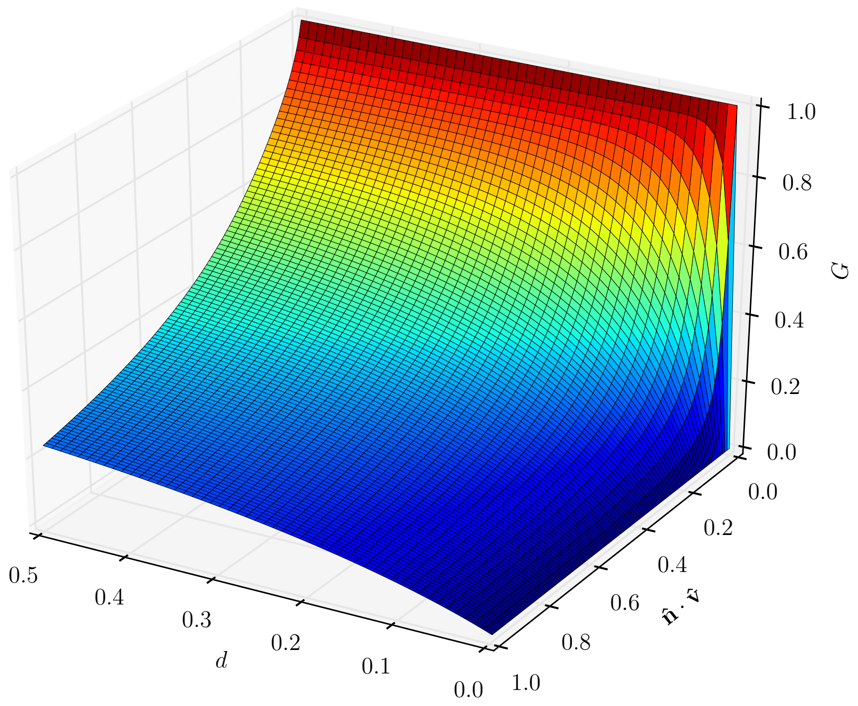
Figure 4: Plots of $P_s(\hat{\mathbf{u}})$ and $\widetilde{P}_s(\hat{\mathbf{u}})$ with $\alpha = 0.75$.

C $\widetilde{G}$ CONSTANTS AND EVALUATION CODE

```
float FuzzGApprox(float NdotV, float density)
{
    // Clamp to the domain used for curve fitting

    const float densityMax = 0.5f;
    density = min(density, densityMax);

    const float c[] = { -0.94634516f, 0.41691235f, 0.70663116f, -0.26154413f };

    return (density * (1.0f + NdotV * (c[0] + NdotV * c[1]))) /
               (density + NdotV * (c[2] + NdotV * NdotV * c[3]));
}
```

Listing 3: HLSL implementation of Equation 3.

D PLOTS OF G AND $\widetilde{G}$



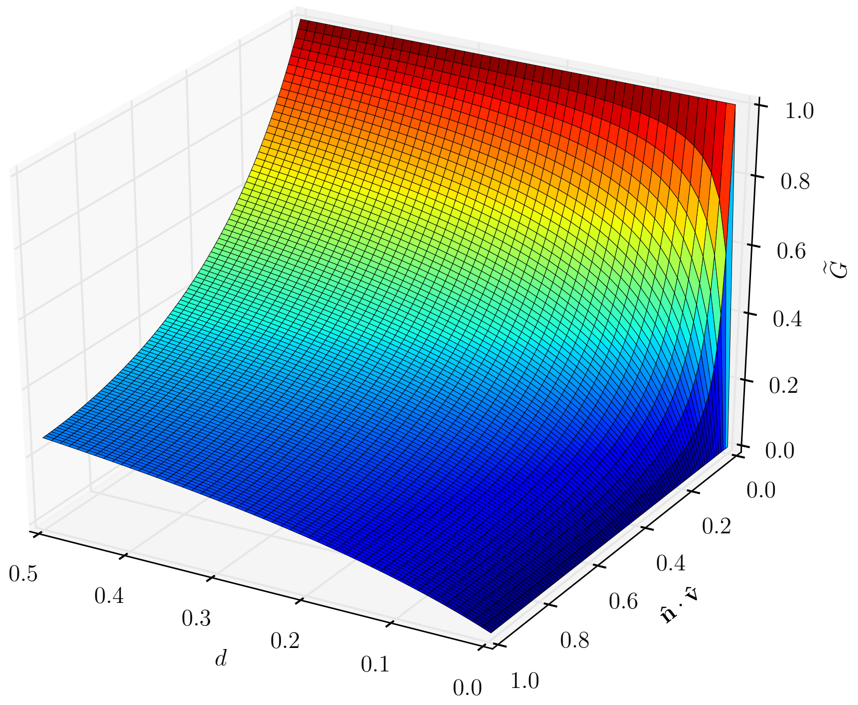


Figure 5: Plots of $G(d, \hat{\mathbf{v}})$ (top) and $\widetilde{G}(d, \hat{\mathbf{v}})$ (bottom).

E $\widetilde{Q}$ CONSTANTS AND EVALUATION CODE

```
float FuzzAveragePenetration(float density)
{
    // Clamp to the domain used for curve fitting

    const float densityMax = 0.5f;
    density = min(density, densityMax);

    const float c[] = { -1.7280754f, 1.2661427f };

    return 1.0f + density * (c[0] + density * c[1]);
}
```

Listing 4: HLSL implementation of Equation 4.

F PLOT OF Q AND $\widetilde{Q}$

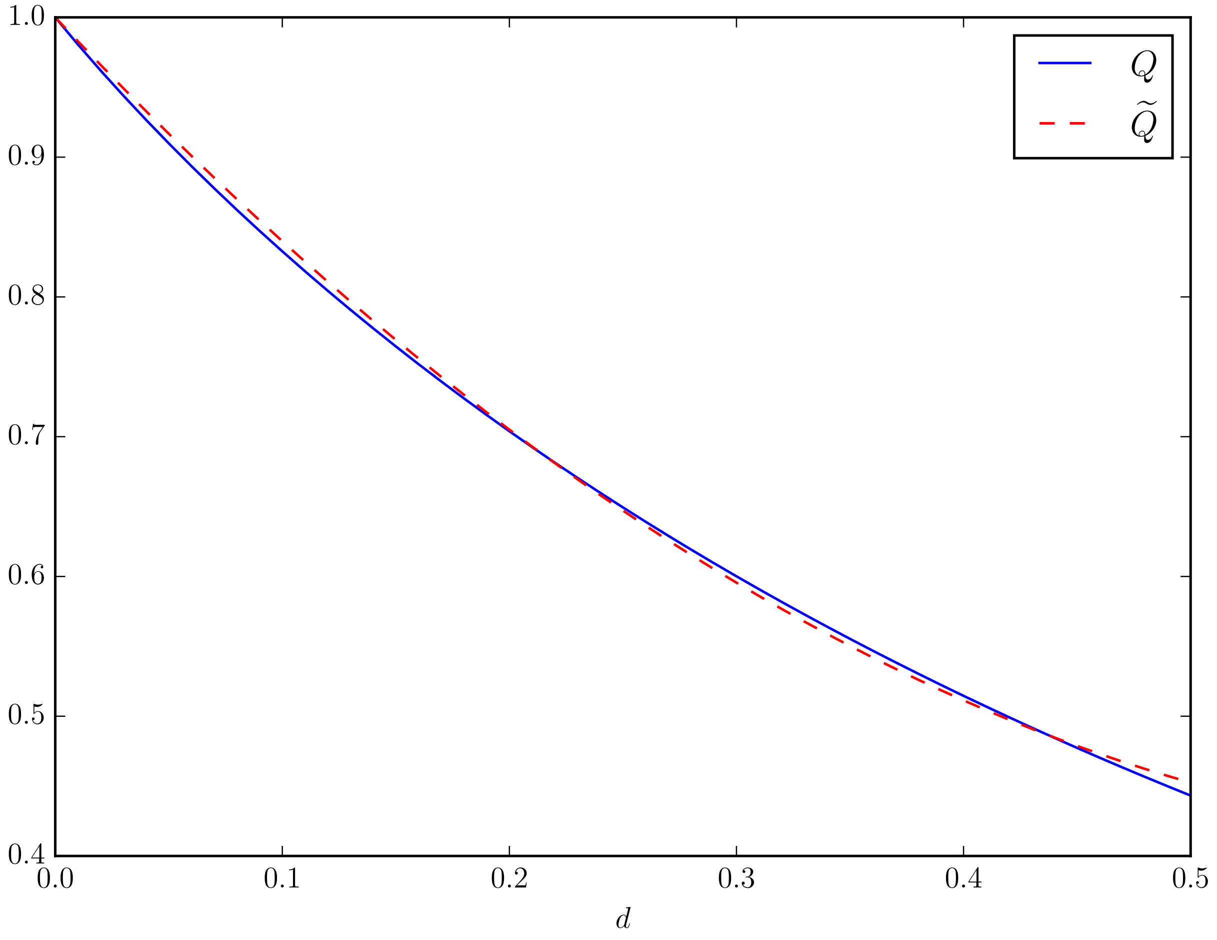


Figure 6: Plot of $Q(d)$ and $\widetilde{Q}(d)$.

G $\widetilde{R}$ CONSTANTS AND EVALUATION CODE

```
float FuzzAveragePs(float alpha, float thetaF)
{
    static const float c[17] =
        {
            -0.26805773f,
            0.30887989f,
            0.077007705f,
            -0.12175263f,
            -0.14202922f,
            0.7331539f,
            0.010009714f,
            -0.25925012f,
            -0.023091515f,
            -0.31883099f,
            0.25857198f,
            -0.0031055288f,
            0.20038139f,
            -0.28705017f,
            -0.37659157f,
            -0.44286302f,
            -0.35584712f,
        };

    // Clamp to the domain used for curve fitting

    float alpha = GSggxAlphaFromFuzzSpread(uSpread);
    thetaF = max(0.0f, min(0.5f * pi, thetaF));

    static const float s_alphaMin = 0.15f;
    alpha = max(s_alphaMin, min(1.0f, alpha));

    float sinThetaF = sinf(thetaF);
    float cosThetaF = cosf(thetaF);

    // beta, gamma, delta

    float b = 1.0f - alpha;
    float g = cosThetaF;
    float d = sinThetaF;

    float b2 = b * b;
    float b3 = b2 * b;

    float g2 = g * g;
    float g3 = g2 * g;

    float d2 = d * d;
    float d3 = d2 * d;

    return 0.5f + b * (c[0] + c[1] * b + c[2] * g + c[3] * d + c[4] * b2 + c[5] * g2 +
                       c[6] * b * g + c[7] * b * d + c[8] * b3 + c[9] * g3 + c[10] * d3 +
                       c[11] * b2 * g + c[12] * b2 * d + c[13] * b * g2) /
        (1.0f + c[14] * b + c[15] * g + c[16] * d);
}
```

Listing 5: C++ implementation of Equation 5.

H PLOTS OF R AND $\widetilde{R}$

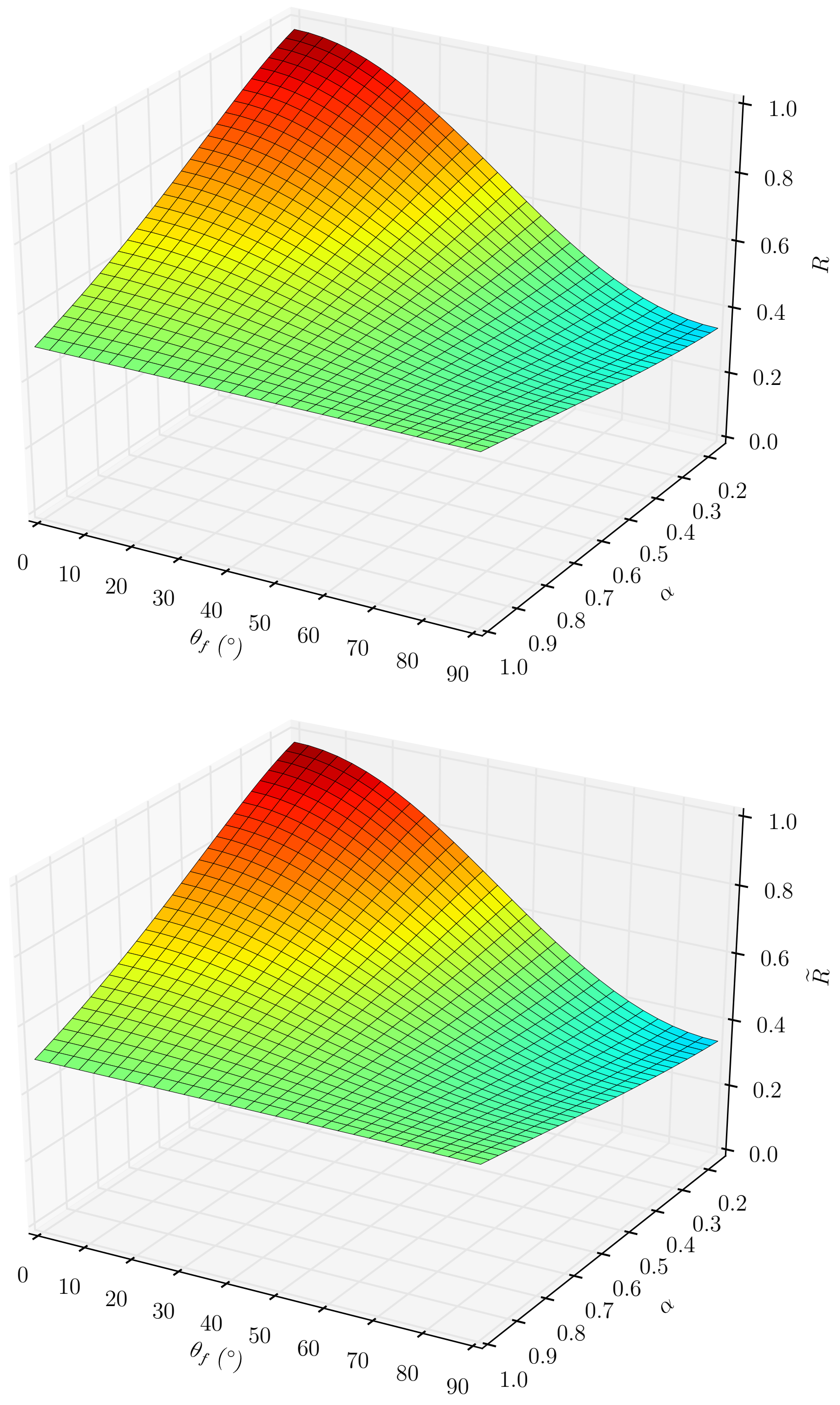


Figure 7: Plots of $R(\theta_f, \alpha)$ (top) and $\widetilde{R}(\theta_f, \alpha)$ (bottom).