

GLASS: Graphics Language Acceleration for Spectral Synthesis

N. R. Hill^{*} and R. H. D. Townsend

Department of Astronomy, University of Wisconsin-Madison, 2535 Sterling Hall, 475 North Charter Street, Madison, WI 53706, USA

Accepted Received

ABSTRACT

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Key words: methods: numerical – stars: variable: general – binaries: eclipsing

1 INTRODUCTION

Modeling the spectrum of a star is a two-step process: calculate the outbound radiation at each point on the stellar surface (the spectral synthesis step), and then sum over the subset of points which are visible to the observer (the disk-integration step). For a spherically symmetric star with uniform surface properties, the disk integration step is trivial; however, for non-spherical stars and/or stars with inhomogeneous surfaces, it can be complex and computationally intensive. In their seminal work on modeling the spectra of close binary systems, Wilson & Devinney (1971) remark that “The most difficult part of the problem is that of summing the flux in the observer’s direction while correctly accounting for horizon and eclipse effects [...] without being able to use the simplified geometry afforded by sphere or ellipsoid models.”

These difficulties are not unique to close binary systems; they are also encountered when modeling contact binaries (e.g., Lucy 1968; Hendry & Mochnacki 1992), and stars with spots (e.g., Vogt et al. 1987; Collier Cameron 1997), pulsations (e.g., Townsend 1997; Schrijvers et al. 1997; De Ridder 2001) and magnetic fields (e.g., Piskunov & Kochukhov 2002). The disk integration approaches adopted in these studies share many similarities, but at the same time each is sufficiently domain-specific that it is difficult to re-purpose

to other types of problem. This duplication of effort provides a motivation to search for a unified technique which can easily be applied to a broad variety of stellar systems.

Here, we present a new method — Graphics Language Acceleration for Stellar Spectra (GLASS) — that leverages the OpenGL and CUDA computer technologies. OpenGL¹ is an industry-standard application programming interface (API) for rendering 2-D and 3-D graphics, which takes advantage of graphics processing unit (GPU) hardware. CUDA is a parallel computing environment and supporting API developed by NVIDIA Corporation to support general-purpose computing on their graphics hardware products. (We provide a brief technical background to both OpenGL and CUDA in Appendix A). In the GLASS method the disk integration is handled via calls to the OpenGL API, while the spectral synthesis uses CUDA’s texture functionality to interpolate in pre-computed grids of intensity spectra.

The following section frames the spectral modeling problem in general terms. Section 3 then describes the GLASS method and introduces GLADOS, an implementation of the method written in the C language. We validate and benchmark GLADOS in Section 4, and then apply it in Section 5 to example problems. Section 6 discusses the GLASS method in the context of other spectral modeling approaches, and Section 7 summarizes the paper.

^{*} E-mail:nhill@astro.wisc.edu

¹ <http://www.opengl.org>

2 PROBLEM FORMULATION

Let $I(\lambda, \mathbf{r}, \hat{\mathbf{s}})$ denote the photospheric specific intensity of radiation at wavelength λ , emitted in a direction $\hat{\mathbf{s}}$ from a point with position \mathbf{r} on the surface of a star (here and throughout, a circumflex accent denotes a unit vector). The flux measured by an observer at a distance D from the star (assumed to be large) is expressed as

$$F_{\text{obs}}(\lambda) = \frac{1}{D^2} \int_{\text{vis}} I(\lambda, \mathbf{r}, \hat{\mathbf{s}}_{\text{obs}}) \mu dA, \quad (1)$$

where $\hat{\mathbf{s}}_{\text{obs}}$ is the vector pointing toward the observer, $\mu \equiv \hat{\mathbf{s}}_{\text{obs}} \cdot \hat{\mathbf{n}}$ is the projection of the local surface normal vector $\hat{\mathbf{n}}$ onto this vector, and dA is the surface area element. The integral is evaluated over the portion of the surface visible to the observer, and composes the disk-integration step mentioned in the Introduction; evaluating I in the integrand likewise constitutes the spectral synthesis step.

We assume that the stellar photosphere can be approximated locally by a plane parallel atmosphere in uniform motion with velocity \mathbf{v} . This is reasonable if the photosphere is geometrically thin, with a scale height much smaller than its radius of curvature; and if it doesn't contain any dynamical substructures such as shocks. Then, equation (1) can be recast as

$$F_{\text{obs}}(\lambda) = \frac{1}{D^2} \int_{\text{vis}} I_{\text{pp}}(\lambda_0, T_{\text{eff}}, g_{\text{eff}}, \mu) \mu dA, \quad (2)$$

where $I_{\text{pp}}(\lambda_0, T_{\text{eff}}, g_{\text{eff}}, \mu)$ is the specific intensity at rest wavelength $\lambda_0 = \lambda(1 + \mathbf{v} \cdot \hat{\mathbf{s}}_{\text{obs}}/c)$ emitted by a static plane-parallel atmosphere with effective temperature T_{eff} and effective gravity g_{eff} at an angle $\theta = \cos^{-1} \mu$ to the vertical. All of these parameters are implicit functions of position \mathbf{r} ; the intensity may also depend on additional parameters, e.g. chemical abundance, but for simplicity we neglect this here.

In the case of a uniform, spherical star with radius R , where the effective temperature and gravity are independent of position on the surface, the flux simplifies to

$$F_{\text{obs}}(\lambda) = \frac{R^2}{D^2} \left[2\pi \int_0^1 I_{\text{pp}}(\lambda_0, T_{\text{eff}}, g_{\text{eff}}, \mu) \mu d\mu \right]. \quad (3)$$

The quantity in brackets can be recognized as the surface flux of the plane-parallel atmosphere, a quantity often published in tables (e.g., Hauschildt et al. 1999; Castelli & Kurucz 2003; Lanz & Hubeny 2003, 2007). For the more general cases which are the focus of the present paper, however, the full form given in equation (2) must be used.

3 METHOD DESCRIPTION

3.1 Overview

The GLASS method evaluates the observed flux following equation (2). The surface geometry of the star is represented using a triangle-based mesh, comprising a set of vertices situated on the surface together with connectivity information which groups vertices together to form triangular faces. Each face is defined by three vertices, and has three adjacent faces with which it shares a common edge. The physical state of the surface is established by defining a set of relevant scalar properties (e.g., T_{eff} , g_{eff}) and vector properties (e.g., $\hat{\mathbf{n}}$, \mathbf{v})

on a per-face or per-vertex basis, or via separate latitude-longitude maps; these alternatives are discussed further below.

To calculate the flux, the mesh is first projected onto the sky plane (perpendicular to $\hat{\mathbf{s}}_{\text{obs}}$) and rasterized to create a “property image”. This is a two-dimensional ($N_i \times N_j$) array of pixels, with each pixel storing property values derived from the part of the sky-projected mesh it covers. Then, the flux is evaluated using a discrete form of equation (2),

$$F_{\text{obs}}(\lambda) = \frac{dA_{\text{p}}}{D^2} \sum_{i=1}^{N_i} \sum_{j=1}^{N_j} I_{\text{pp}}(\lambda_0^{i,j}, T_{\text{eff}}^{i,j}, g_{\text{eff}}^{i,j}, \mu^{i,j}), \quad (4)$$

where dA_{p} is the pixel area, and the i, j superscripts indicate properties of the pixel at integer array coordinates (i, j) . For pixels which do not cover any part of the sky-projected mesh (so-called background pixels), the properties are undefined and the corresponding intensity term in the summation is set to zero.

The sky-projection and rasterization are handled via calls to the OpenGL API. The specific intensity is interpolated in pre-computed grids, using the texture interpolation functionality exposed through the CUDA API. The following sections describe these steps in greater detail.

3.2 Mesh Generation

3.2.1 Spherical Star

A mesh for a single, spherical star with radius R is created by repeated subdivision of an icosahedron. The initial icosahedron is defined so that all points lie at distance R from the origin. During a subdivision, each face is divided into four equal-area faces; this creates a new vertex halfway along each edge. Then, the new vertices are displaced radially so that they also lie a distance R from the origin. After ν such subdivisions, the mesh contains

$$N_{\text{f}} = 4^\nu 20 \quad (5)$$

faces, and

$$N_{\text{v}} = 2^\nu 12 + 2^{\nu-1} (2^\nu - 1) 20 - 2^{\nu+1} + 2 \quad (6)$$

vertices; the number of edges N_{e} follows from Euler's formula as $N_{\text{e}} = N_{\text{f}} + N_{\text{v}} - 2$. Figure 1 shows an example of meshes generated using this approach, for $\nu = 0, 3$ and 6. Evidently, toward larger ν the mesh provides a progressively better approximation to a sphere.

3.2.2 Non-Spherical Star

For rapidly rotating single stars, the departure from sphericity due to the centrifugal force must be taken into account. The algorithm above is modified so that the vertices, in the initial icosahedron and after each subdivision, are moved radially to lie a distance $R(\theta)$ from the origin, where θ is the colatitude in the spherical polar coordinate system aligned with the star's rotation axis. In the Roche approximation where the star is treated as a point mass,

$$R(\theta) = \frac{3}{\omega \sin \theta} \cos \left[\frac{\pi + \cos^{-1}(\omega \sin \theta)}{3} \right] R_{\text{p}} \quad (7)$$

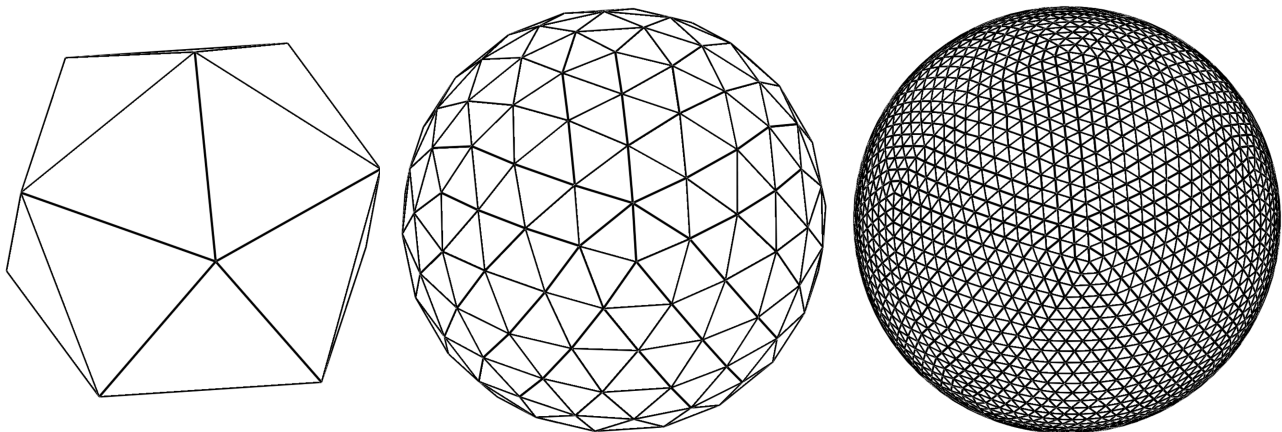


Figure 1. Subdivided icosahedron meshes with (left-to-right) $\nu = 0, 3$ and 6 subdivisions.

(e.g. Cranmer 1996); here,

$$\omega = \Omega \sqrt{\frac{27R_p}{8GM}} \quad (8)$$

is the ratio between the rotation angular velocity of the star Ω and the corresponding critical angular velocity, and R_p and M are the polar radius and mass of the star, respectively.

3.2.3 Binary Stars

In tidally locked binary systems, the surfaces of both stellar components are assumed to lie on Roche equipotential surfaces (Kopal 1959). A general approach to creating meshes for these is to apply an isosurface extraction algorithm. The marching cubes algorithm Lorensen & Cline (1987) is the most widely used, but the regularized marching tetrahedra (RMT) algorithm described by Treece et al. (1999) generally produces higher-quality meshes. In the RMT algorithm, the scalar isosurface function (in the present case, the Roche potential) is first sampled on a body-centered cubic lattice with a unit cell of side length Δ . Tetrahedra are then formed by considering groups of four neighboring grid points at a time. By examining the value of the scalar function at these points, it can quickly be determined if the isosurface intersects the tetrahedron, and if so, the mesh triangles needed to represent this intersection.

Figure 2 illustrates meshes generated by the RMT algorithm for a model of the contact binary W UMa (see Section 5.1). The three meshes in the figure correspond to $N_\Delta = 32, 48$ and 64 , where the resolution parameter N_Δ sets the cell length via $\Delta = d_1/N_\Delta$, and d_1 is the distance from the center of the primary to the L1 point. As N_Δ increases the mesh quickly approaches a very good approximation to the Roche equipotential.

3.3 Rasterization

Mesh rasterization proceeds face-by-face, assigning property values to each of the image pixels whose center falls inside the triangular outline formed by a face's sky projection. The

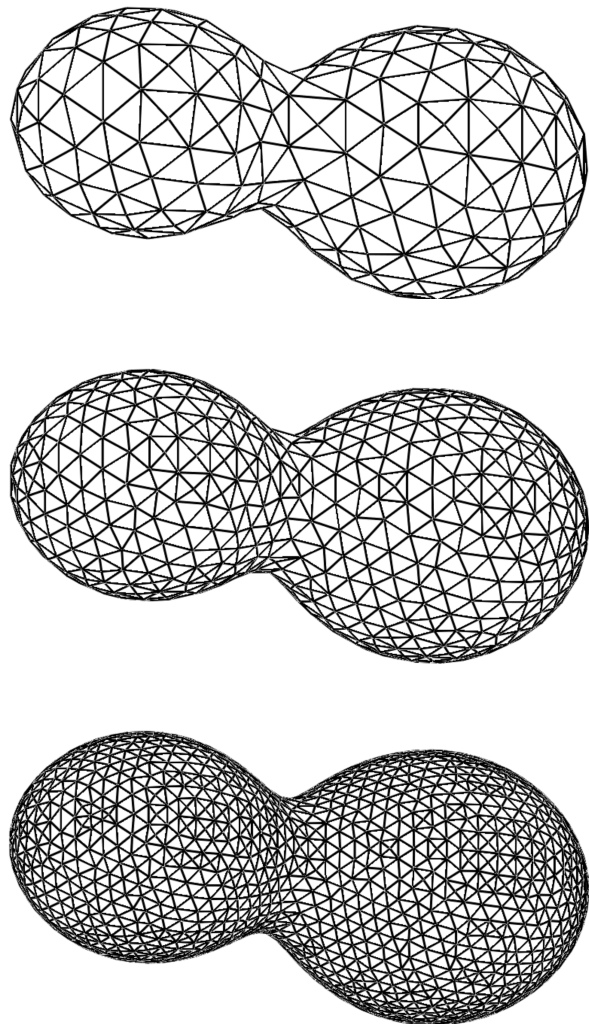


Figure 2. Meshes representing the contact binary system W UMa (see Section 5.1), generated by the RMT algorithm with (top-to-bottom) $N_\Delta = 32, 48$ and 64 .

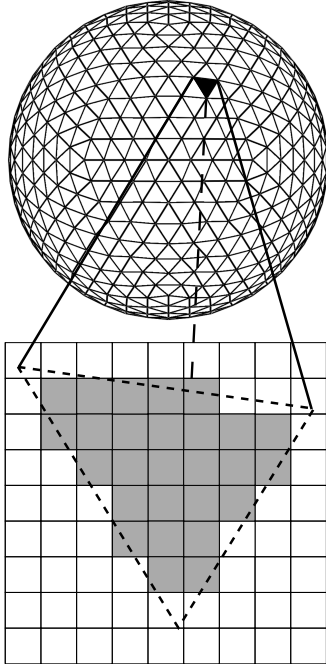


Figure 3. Illustration of how a single face from a subdivided icosahedron mesh (top) is rasterized using flat shading (bottom): all pixels within the triangular outline are assigned the same property value as the face

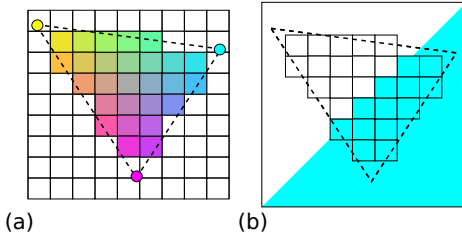


Figure 4. Illustration of smooth shading (left) and texture mapping (right), for rasterization of the same face shown in Figure 3.

OpenGL API offers a variety of methods to calculate these property values from the corresponding values on the mesh.

With ‘flat shading’, mesh properties are defined on a per-face basis, and all pixels within the triangular outline are assigned the same property values as the face; this process is illustrated in Figure 3. With ‘smooth shading’, mesh properties are defined on a per-vertex basis, and each pixel within the outline is assigned property values by bilinear interpolation between the vertex values. Finally, with ‘texture mapping’ mesh properties are defined indirectly by associating a coordinate pair with every vertex. Each pixel within the outline is then assigned property values by interpolation in separate 2-D arrays known as texture maps, at locations given by the interpolated coordinate pair of the pixel. Smooth shading and texture mapping are illustrated in Figure 4.

Hidden surface removal (the “horizon and eclipse effects” from the quotation in Section 1) is handled by pairing the property image with a separate ‘depth buffer’ array hav-

ing the same $N_i \times N_j$ dimensions. When a pixel is considered for assignment, its distance from the observer (as interpolated from the vertex distances) is first compared to the value stored in the corresponding pixel of the depth buffer. If the former is smaller than the latter, then the assignment proceeds and the depth buffer is updated with the new distance value.

3.4 Spectral Synthesis

The plane parallel specific intensity I_{pp} is synthesized by interpolating in a pre-computed grid of specific intensity spectra. GLADOS uses piecewise linear interpolation in four-dimensional (T_{eff} , $\log g$, λ_0 and μ) grids. In the following sections, the adopted intensity grid is computed with the SYNSPEC code (Lanz & Hubeny 2003) from a corresponding solar-composition grid of local thermodynamic equilibrium (LTE) model atmospheres (Castelli & Kurucz 2003). The grid covers effective temperatures $3,500 \text{ K} \leq T_{\text{eff}} \leq 50,000 \text{ K}$ and surface gravities from $\log g = 5$ down toward the Eddington limit. Individual intensity spectra span the wavelength range $880\text{--}7,500 \text{ \AA}$ at a resolution $R = 50,000$, and are sampled with 21 uniformly spaced points over the $0 \leq \mu \leq 1$ interval.

3.5 Implementation Details

We now discuss some of the more technical aspects of the GLASS method, as implemented in the GLADOS code. OpenGL context creation, which falls outside the purview of the OpenGL API, is handled using the open source, cross-platform GLFW library². Mesh data, including vertex coordinates and properties, are stored in OpenGL vertex buffer objects. Transformations (rotations, translations) to position the mesh prior to sky projection are handled using a GLSL vertex shader (see Appendix A1), while the flat shading, smooth shading and texture mapping options for rasterization are handled using GLSL fragment shaders. A property image (Section 3.1) comprising n_s scalar properties and n_v vector properties is represented via an OpenGL framebuffer object (FBO) containing multiple renderbuffers:

- One renderbuffer with format `GL_DEPTH_COMPONENT32F`, attached to the FBO’s `GL_DEPTH_ATTACHMENT` point; this serves as the depth buffer.
- n_s renderbuffers with format `GL_R32F`, attached to the FBO’s `GL_COLOR_ATTACHMENT i` points ($i = 0, \dots, n_s - 1$).
- n_v renderbuffers with format `GL_RGB32F`, attached to the FBO’s `GL_COLOR_ATTACHMENT i` points ($i = n_s, \dots, n_s + n_v - 1$).

The `GL_RGB32F` format (ordinarily used to represent red, green and blue color channels) is adopted for vector properties because it can store three floating-point values per renderbuffer pixel, corresponding to the three components of a vector.

When many wavelength points $\{\lambda_k\}$ ($k = 1, \dots, N_\lambda$) are required in the observed spectrum $F_{\text{obs}}(\lambda)$, the intensity interpolation can be the most computationally expensive part

² <http://www.glfw.org/>

of the spectral modeling process. Therefore, GLADOS implements the interpolation following the approach described by Townsend et al. (2011). First, the intensity grid is loaded into texture memory on the GPU (a type of cached device memory optimized for spatial locality). Then, making use of standard OpenGL-CUDA interoperability functionality, property image pixels from the rasterization are passed as a stream of $(\lambda_0, T_{\text{eff}}, g_{\text{eff}}, \mu)$ tuples to a CUDA kernel. For each tuple in this stream, the kernel interpolates the specific intensity in the grid using multiple calls to CUDA's `tex2D()` function. (Although this function supports only bilinear interpolation in two-dimensional tables, higher-dimensional interpolations can be broken into a weighted sum of two-dimensional interpolations). The interpolated intensity is then added to the flux spectrum via equation (4).

4 VALIDATION

As an initial validation of the GLASS method described in the preceding section, we use the GLADOS code to model the optical spectrum of the Sun. The solar surface is represented using a subdivided icosahedron mesh with uniform $T_{\text{eff}} = 5,778 \text{ K}$ and $\log g = 4.44$, and all effects of rotation are neglected. Spectra are calculated over the interval $4,000 \text{ \AA} - 7,000 \text{ \AA}$ at a resolution $R = 50,000$, for eight levels of mesh subdivision ($\nu = 0, \dots, 7$) and 23 choices of property image dimensions: $N_i = N_j = N = 32$ up to 512 in increments of 32, and from there up to 960 in increments of 64. The (square) images have a sky-projected side length $2.4 R_{\odot}$, leading to pixel areas $dA_p = (2.4 R_{\odot})^2 / (N_i N_j)^2$. For each of these spectra, we calculate the root-mean-square (RMS) relative error

$$\epsilon = \left\{ \frac{1}{N_{\lambda}} \sum_{k=1}^{N_{\lambda}} \left[\frac{F_{\text{obs}}(\lambda_k) - F_{\odot}(\lambda_k)}{F_{\odot}(\lambda_k)} \right]^2 \right\}^{1/2} \quad (9)$$

with respect to a reference solar spectrum $F_{\odot}(\lambda)$ calculated (using the same intensity grids) via equation (3).

Figure 5 shows the results of this comparison exercise, plotting ϵ as a function of N for the eight levels of mesh subdivision. The three panels correspond to the different rasterization approaches described in Section (3.3): flat shading, smooth shading and texture mapping (since T_{eff} and $\log g$ are uniform across the mesh, it is only the projected normal μ which is affected by the choice of approach). All three panels show generally similar behavior: ϵ converges to constant limiting value ϵ_{∞} as N is increased, and ϵ_{∞} itself depends inversely on ν .

This behavior is a consequence of the three principal factors which contribute toward ϵ . First, the rasterization process samples the sky-projected mesh with a finite resolution, and can miss details on a scale smaller than the pixel size. Second, the mesh itself is an approximation to the true stellar geometry, leading for instance to an underestimation of the projected area of the star when a coarse mesh is used (see Hendry & Mochnacki 1992, their section 2.3, for a discussion of this effect). Finally, in the CUDA-based texture interpolation used for the spectral synthesis, the weights are calculated using 9-bit fixed precision arithmetic, which has the effect of introducing numerical noise at a level of one part in $2^9 = 512$ (Townsend et al. 2011).

The rasterization errors are minimized by choosing a

sufficiently large N ; this is seen in Figure 5 by the convergence of each curve to its limiting value ϵ_{∞} . The mesh errors can likewise be minimized by choosing a sufficiently large ν ; this is seen in the figure by the decrease of ϵ_{∞} as ν is increased. However, the numerical noise arising in the interpolation means that no choice of N and/or ν can reduce the RMS deviation below the floor $\epsilon_{\infty} \approx 3 \times 10^{-3}$. Given that this floor falls comfortably beneath the noise level of most observations that GLASS spectra will be compared against, in practice this limitation should not be a problem.

Comparing the different panels in the figure reveals only minor differences between the alternative rasterization approaches. Flat shading achieves slightly smaller ϵ than smooth shading, because its rasterization errors partly cancel its mesh errors. However, as we shall illustrate below in Section 5.2, texture mapping in general produces much better results than flat or smooth shading when modeling stars with non-uniform surface properties.

5 EXAMPLE CALCULATIONS

5.1 Contact Binary

To illustrate GLADOS's ability to model stars with complex surface geometries, we apply the code to calculate light curves of W UMa (HD 83950; F8Vp + F8Vp), the archetype for low-mass contact eclipsing binary systems. Stellar and orbital parameters are adopted from the '1989' column of table 3 of Rucinski et al. (1993). The mesh representing the single equipotential surface of the system is constructed using the RMT algorithm described in Section 3.2.3 with $N_{\Delta} = 64$, and is illustrated in the lower panel of Figure 2. The effective temperature of individual vertices is set as

$$T_{\text{eff}} = C g_{\text{eff}}^{\beta}, \quad (10)$$

where β is a gravity-darkening exponent and C a constant chosen so that the bolometric luminosity of the system (calculated by integrating σT_{eff}^4 over the surface) is $2.16 L_{\odot}$. We adopt $\beta = 0.08$ obtained for convective envelopes by Lucy (1967). For simplicity, any reflection effects due to irradiation of one part of the surface by another are neglected (although see Townsend, Hill, et al., in preparation, for a demonstration of how these effects can be incorporated).

At 128 uniformly distributed phases φ around one orbit, GLADOS rasterizes the mesh with smooth shading and a pixel area $dA_p = (6 R_{\odot})^2 / 512^2$, and synthesizes spectra at a resolution $R = 500$. The spectra are then convolved with the passband function tabulations by Bessell (1990) to yield U , B and V light curves. These light curves, plotted in the left-hand panel of Figure 6, exhibit the general characteristics of the W UMa class of eclipsing binary: a smooth variation in brightness, with broad maxima and narrow minima.

The right-hand panel of Figure 6 shows images of the V -band flux F_V at four orbital phases. These are constructed by synthesizing a spectrum for each pixel in the property images, to form a hyperspectral datacube with axes of position-position-wavelength. The datacube is then convolved along its wavelength axis with the V passband function to yield a flux image. The images reveal that the deeper light minimum (by convention, placed at phase $\varphi = 0$) occurs when the primary star is eclipsed by the secondary. Although the primary

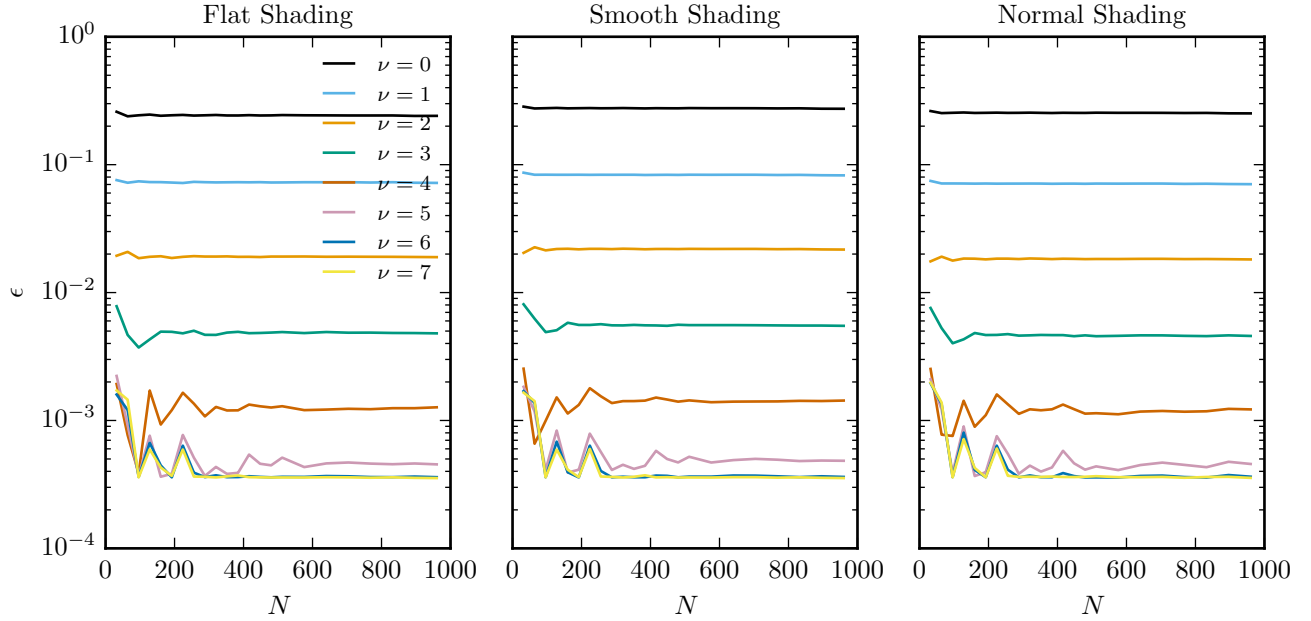


Figure 5. The RMS relative deviation ϵ of model solar spectra from the reference spectrum, plotted as a function of the property image dimension N . Each curve corresponds to a different number of mesh subdivisions ν , and the three panels show the three different approaches.

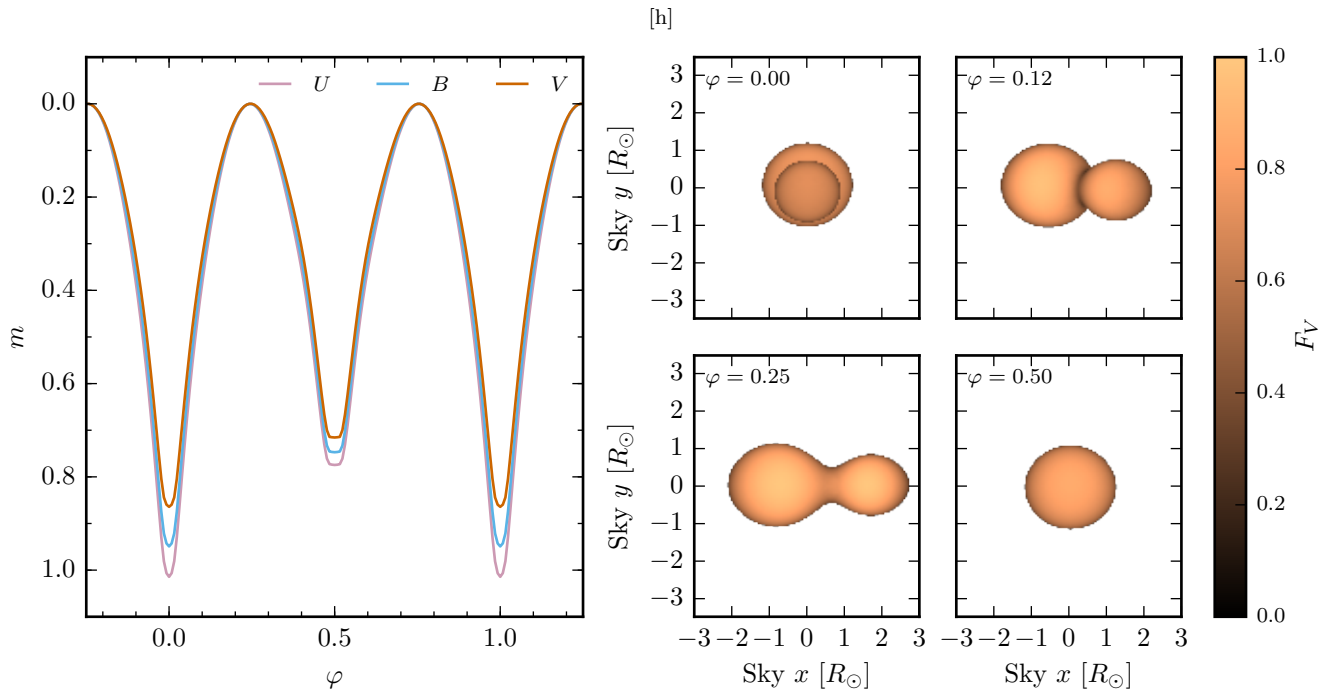


Figure 6. Left: Light curves (in magnitudes m) of the W UMa model in the U , B and V bands. Each curve is normalized to $m = 0$ at quadrature phases ($\varphi = 0.25$ and $\varphi = 0.75$). Right: images of the V -band flux at four selected orbital phases, normalized to a maximum of unity. Background pixels are shown as white.

is a little hotter than the secondary, due to its stronger surface gravity, it is not this temperature difference but rather the effects of limb darkening — seen quite clearly in the figure — which cause the deeper minimum to occur at primary eclipse.

In the classification scheme introduced by Binnendijk (1970), the light curves shown in Fig. 6 are ‘A’-type. In reality, W UMa shows a ‘W-type’ light curve, with the deeper light minimum occurring when the secondary is eclipsed by the primary. This discrepancy is a well-known limitation of models (like ours) which assume a barotropic photosphere which follows a simple gravity darkening law. Attempts to resolve the discrepancy have for instance invoked spots which lower T_{eff} across the surface of the primary, and departures from barotropicity which raise T_{eff} across the surface of the secondary (see Linnell 1991, and references therein).

5.2 Spotted Sun

To illustrate GLADOS’s ability to model stars with inhomogeneous surface properties, we apply the code to calculate light curves of a solar model with a single, circular equatorial spot spanning 0.2% of the total surface area. As in Section 4, the surface is represented using a subdivided icosahedron mesh; the effective temperature outside (inside) the spot boundaries is $T_{\text{eff}} = 5,778\text{ K}$ ($4,622\text{ K}$), and $\log g = 4.44$ everywhere.

Light curves over one rotation cycle are synthesized as in Section 5.1, but for two levels of mesh subdivision ($\nu = 3, 6$) and using either smooth shading or texture mapping for the rasterization. To eliminate the small variability which arises from the rotation of the mesh, we normalize calculated fluxes by the corresponding flux of an unspotted model with otherwise identical parameters. The left-hand panel of Fig. 7 shows the resulting V -band curves for the four possible parameter combinations. Three combinations show good agreement with each other, but one ($\nu = 3$, smooth shading) clearly over-predicts the depth of the light minimum by a factor ≈ 2 .

The reason for this discrepancy can be seen in the right-hand panel of the figure, which shows V -band flux images at rotation phase $\varphi = 0$ for the four parameter combinations (as in Section 5.1, the images are calculated by convolving a hyperspectral datacube with the V passband function). In the $\nu = 3$ / smooth shading case the limited resolution of the surface mesh results in an spot which is hexagonal in shape and larger than the spots shown in the other images. The corresponding $\nu = 3$ / texture mapping case does not suffer from this problem, even though it uses the same mesh, because the T_{eff} texture map has a resolution (600 points in latitude, 1,200 in longitude) more than sufficient to accurately reproduce the spot. This is a key strength of texture mapping: the ability to represent detail at scales smaller than the resolution of the surface mesh.

5.3 Pulsating Star

To illustrate GLADOS’s ability to model stars with complex surface velocity fields, we apply the code to calculate time-series spectra for the pulsating Be star ω CMa

(HD 56139; B3IVe). Stellar parameters are adopted from table 1 of Maintz et al. (2003). The surface is represented using a subdivided icosahedron mesh with $\nu = 4$; for simplicity, the deformation and gravity darkening due to rotation are neglected. The star’s single $\ell = 2$ sectoral pulsation mode is modeled within the traditional approximation, with surface velocity fields given by Hough functions (e.g., Townsend 2003). Temperature perturbations are calculated under the adiabatic approximation.

Figure 8 illustrates the resulting time-series spectra for three spectral lines commonly used in studies of pulsating B stars: Mg II 4481Å, Si III 4553Å and He I 4713Å. The upper panels plot the departure from a reference line profile, as a function of velocity v and pulsation phase φ . The lower panels show the reference profile (taken to be the profile of a non-pulsating model), together with the instantaneous line profiles at selected phases.

The behavior seen in the figure broadly mirrors that found by Maintz et al. (2003): bumps and dips of excess absorption and emission migrate from blue to red across the line profiles. The details vary from profile to profile, due to each profile’s unique shape and sensitivity toward temperature perturbations. The continuum variability on either side of the Si III line is due to blending with Fe II lines. A detailed comparison against figure 1 of Maintz et al. (2003) reveals small differences, but these can reasonably be attributed to our neglect of centrifugal effects, and to our use of an LTE intensity grid.

6 DISCUSSION

To provide some context for evaluating the GLASS method, we now compare and contrast it against some of the approaches to modeling binary systems which appear in the existing literature.

In his pioneering model of W UMa, Lucy (1968) used an approach analogous to ‘ray casting’ in computer graphics (e.g., Roth 1982). A set of rays, arranged on a rectangular grid, is cast from the observer toward the Roche equipotential surface. The intersection of each ray with this surface is found using Newton-Raphson iteration; the surface properties at this intersection are then calculated and used to synthesize an observer-directed intensity. Summing over the intensity associated with each ray results in a disk-integrated spectrum for the system. There are strong parallels between this approach and the GLASS method, as can be appreciated by comparing equation 9 of Lucy (1968) against our equation (4). However, a key difference is that ray casting is significantly more expensive than rasterization, due to the Newton-Raphson iteration.

Wilson & Devinney (1971) introduced perhaps the most widely adopted approach to modeling binary systems. The surfaces of the components are sampled by a set of points, distributed so that each represents approximately the same fraction of the total surface area. The disk-integrated spectrum is calculated by summing the observer-directed intensity from each point visible to the observer. Visibilities are determined by examining whether a point from the background component falls inside or outside the silhouette of the foreground component. A disadvantage of this approach, compared to Lucy’s ray-casting approach and the GLASS

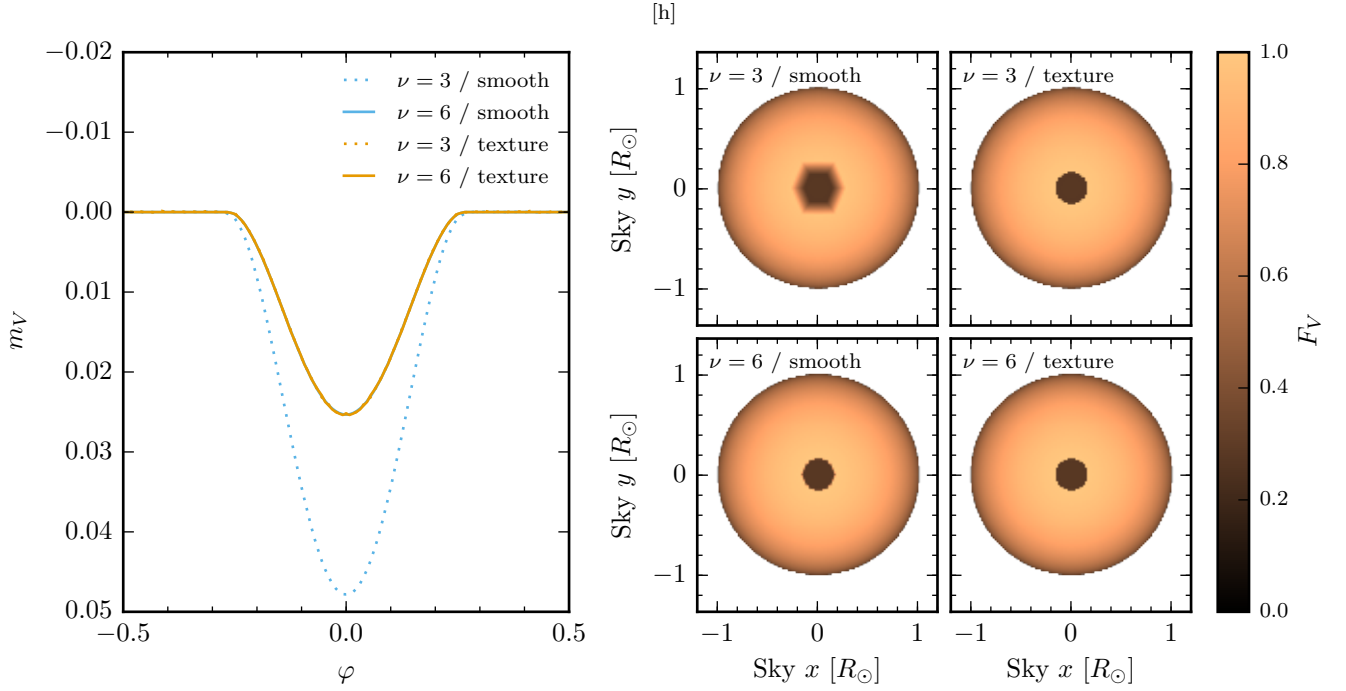


Figure 7. Left: V-band light curves (in magnitudes m_V) of the spotted solar model, for the four choices of subdivision parameter ν and rasterization approach. Each curve is normalized to $m = 0$ when the spot is on the far side ($\varphi \approx 0.5$). The $\nu = 6$ / smooth and $\nu = 3$ / texture curves lie under the $\nu = 6$ / texture curve. Right: images of the V-band flux at phase $\varphi = 0$ for the four choices, normalized to a maximum of unity. Sky pixels are shown in white.

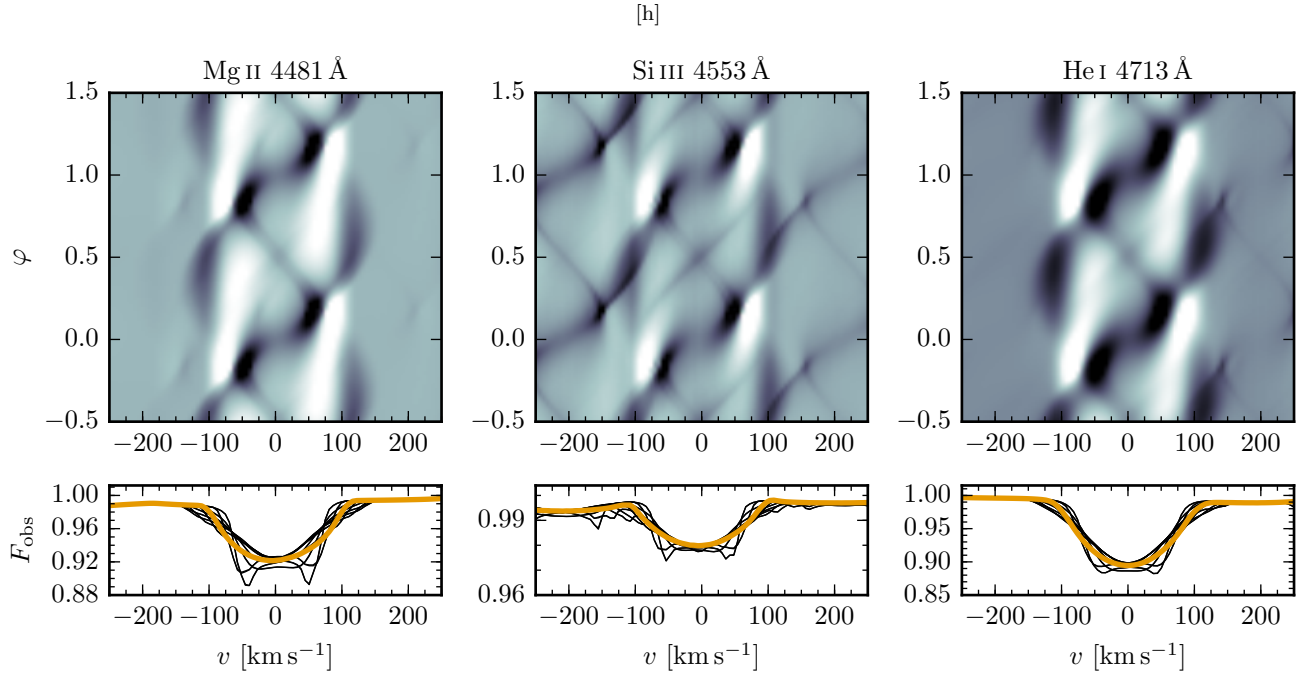


Figure 8. Top: departure from a reference profile, plotted as a function of Doppler velocity v (relative to line center) and pulsation phase φ , for the Mg II 4481 Å, Si III 4553 Å and He I 4713 Å line profiles of the ω CMa model. Black corresponds to maximum excess absorption, and white to maximum excess emission. Bottom: the reference line profile (thick orange line), together with snapshot profiles at eight uniformly-spaced phases (thin black lines). All profiles are rectified by dividing by the continuum flux.

method, is that the sampling points are distributed unevenly over the *projected* surfaces of each component, with a relative overdensity at the limb and an underdensity at the disk center.

In their GDDSYN code, Hendry & Mochnecki (1992) adopted triangle-based meshes to represent the stellar surfaces of binary systems. These are essentially the same as in the GLASS method, although properties can only be specified on a per-face basis. For contact binaries GDDSYN generates meshes using a pair of subdivided icosahedra, truncated and joined to form a throat. Visibilities are determined using a similar silhouette approach to Wilson & Devinney (1971); since the limb of the foreground component is a polygon, the determination can be implemented analytically using a polygon-polygon overlap algorithm. The principal limitation of the GDDSYN approach is the restriction to per-face properties; this means that finely subdivided meshes are required to represent detail at small scales (cf. Section 5.2).

Most recently, the PHOEBE (PHysics Of Eclipsing BinariEs) project³ aims to create a general purpose tool for modeling and analysis of all classes of binary system (at the time of writing, the code is still in the alpha phase of its release cycle). PHOEBE uses a similar approach to GDDSYN, but supports local mesh subdivision to represent small-scale detail. The code also provides a richly featured framework for automated binary-system parameter optimization, controllable through a Python-based API, and can easily be adapted to other problems such as spotted stars.

The significant difference between the GLASS method and these other approaches is GLASS's reliance on established APIs (OpenGL and CUDA) for most of the computationally complex and intensive parts of spectral modeling calculations. This confers a number of important benefits:

- Source code is greatly simplified. This makes the code easier to maintain and further develop, in turn prolonging its useful lifetime.
- Quantitative visualizations (cf. Figures 6 and 7) are trivial to construct. These facilitate analysis and interpretation of results, but also serve as an important check (problems with a model can often be spotted just by examining an image of it).
- Calculations are accelerated by GPU hardware. This allows generation of hundreds or even thousands of spectra per second, supporting computationally intensive tasks such as parameter optimization.

The downside of using OpenGL and CUDA is that GLADOS is in principle restricted to platforms incorporating NVIDIA GPUs. However, this limitation can easily be lifted by reimplementing the CUDA parts of the code in OpenCL (see Appendix A2). Indeed, with OpenCL and a software implementation of OpenGL⁴, it should be possible for GLADOS to run on platforms which lack any GPU hardware whatsoever.

7 SUMMARY

In this paper we introduce the GLASS method for modeling stellar spectra, and demonstrate its application — via the

GLADOS code — to a models of the contact binary W UMa, the Sun with an equatorial spot, and the pulsating Be star ω CMa. These demonstrations confirm that the method can readily model systems containing one or more non-spherical stars and/or stars with inhomogeneous surfaces.

ACKNOWLEDGMENTS

We acknowledge support from NSF Advanced Technology and Instrumentation grant AST-0904607. This research has made use of NASA's Astrophysics Data System.

APPENDIX A: TECHNICAL BACKGROUND

A1 OpenGL

The OpenGL API was originally developed by Silicon Graphics Inc. (SGI) from their proprietary IrisGL API, to provide a cross-platform, vendor-neutral interface to GPUs. It comprises a library of callable functions plus a set of defined constants, which together allow the specification of 2-D and 3-D scenes in terms of primitives (lines, polygons, etc.), and their subsequent rendering into pixel-based images. Operations such as geometric transformation and hidden-surface removal are handled internally by the library.

Since its initial release as version 1.0 in 1992, the OpenGL API has undergone many revisions to keep pace with hardware advances, under the guidance initially of the OpenGL Architecture Review Board, and since 2006 of the Khronos Group Inc., a non-profit industry consortium; at the time of writing the API stands at version 4.5. The most significant developments in the context of the present paper were the introduction in version 2.0 (2004) of the OpenGL shading language (GLSL); and in version 3.0 (2008) of support for rendering into floating-point images. GLSL is a C-like language used to program a GPU's shaders — dedicated hardware units which can execute arbitrary code to manipulate mesh vertices and image pixels during the rendering process.

A2 CUDA

The CUDA (Compute Unified Device Architecture) platform was developed by NVIDIA Corporation in response to growing interest in general-purpose computing on graphics processing units (GPGPU). Initial attempts at GPGPU focused on mapping non-graphics problems into an equivalent set of graphical tasks, through creative manipulation of APIs like OpenGL. These were largely successful (e.g., Larsen & McAllister 2001; Thompson et al. 2002), but demonstrated a clear need for a less ad hoc approach. This spurred the development of BrookGPU (Buck et al. 2004), a compiler and run-time environment which accesses GPU shaders via a stream processing paradigm. In this paradigm, a well-defined series of operations (the kernel) are applied by the shaders to each element in a typically large homogeneous sequence of data (the stream).

Although no longer under active development, BrookGPU influenced the subsequent creation of a number of GPGPU platforms, including NVIDIA's CUDA, AMD's

³ <http://www.phoebe-project.org/>

⁴ E.g., Mesa 3D: <http://www.mesa3d.org/>

FireStream and the cross-platform Open Compute Language (OpenCL) standard⁵. CUDA was first released in 2006, and has since proven a popular choice for GPGPU in many subfields of astrophysics; recent examples of CUDA-enabled science include *N*-body simulations of planet formation (Grimm & Stadel 2014), hydrodynamical simulations of galaxy interactions (Kulikov 2014), genetic classification of globular clusters (Cavuoti et al. 2014) and photon ray-tracing in relativistic spacetimes (Chan et al. 2013).

REFERENCES

- Bessell M. S., 1990, *PASP*, 102, 1181
- Binnendijk L., 1970, *Vistas in Astronomy*, 12, 217
- Buck I., Foley T., Horn D., Sugerman J., Fatahalian K., Houston M., Hanrahan P., 2004, *ACM Trans. Graph.*, 23, 777
- Castelli F., Kurucz R. L., 2003, in Piskunov N., Weiss W. W., Gray D. F., eds, *Proc. IAU Symp. 210: Modelling of Stellar Atmospheres* p. A20
- Cavuoti S., Garofalo M., Brescia M., Paolillo M., Pescape A., Longo G., Ventre G., 2014, *New A*, 26, 12
- Chan C.-k., Psaltis D., Özel F., 2013, *ApJ*, 777, 13
- Collier Cameron A., 1997, *MNRAS*, 287, 556
- Cranmer S. R., 1996, PhD thesis, Bartol Research Institute, University of Delaware, USA
- De Ridder J., 2001, PhD thesis, Institute of Astronomy, Katholieke Universiteit Leuven, Belgium
- Grimm S. L., Stadel J. G., 2014, *ApJ*, 796, 23
- Hauschildt P. H., Allard F., Baron E., 1999, *ApJ*, 512, 377
- Hendry P. D., Mochnacki S. W., 1992, *ApJ*, 388, 603
- Kopal Z., 1959, *Close binary systems*. Wiley, New York
- Kulikov I., 2014, *ApJS*, 214, 12
- Lanz T., Hubeny I., 2003, *ApJS*, 146, 417
- Lanz T., Hubeny I., 2007, *ApJS*, 169, 83
- Larsen E. S., McAllister D., 2001, in *SC '01: Proc. ACM/IEEE Conference on Supercomputing* ACM, New York, p. 55
- Linnell A. P., 1991, *ApJ*, 374, 307
- Lorensen W. E., Cline H. E., 1987, in *SIGGRAPH '87: Proc. 14th Annual Conference on Computer Graphics and Interactive Techniques* ACM, New York, pp 163–169
- Lucy L. B., 1967, *ZAp*, 65, 89
- Lucy L. B., 1968, *ApJ*, 153, 877
- Maintz M., Rivinius T., Štefl S., Baade D., Wolf B., Townsend R. H. D., 2003, *A&A*, 411, 181
- Piskunov N., Kochukhov O., 2002, *A&A*, 381, 736
- Roth S. D., 1982, *Computer Graphics and Image Processing*, 18, 109
- Rucinski S. M., Lu W.-X., Shi J., 1993, *AJ*, 106, 1174
- Schrijvers C., Telting J. H., Aerts C., Ruymaekers E., Henrichs H. F., 1997, *A&AS*, 121, 343
- Thompson C. J., Hahn S., Oskin M., 2002, in *MICRO 35: Proc. 35th Annual ACM/IEEE International Symposium on Microarchitecture* IEEE Computer Society Press, Los Alamitos, pp 306–317
- Townsend R., Sankaralingam K., Sinclair M. D., 2011, in Mei-Hwu W., ed., *GPU Computing Gems*. Morgan Kaufmann, pp 93–102
- Townsend R. H. D., 1997, *MNRAS*, 284, 839
- Townsend R. H. D., 2003, *MNRAS*, 340, 1020
- Treece G., Prager R., Gee A., 1999, *Computers & Graphics*, 23, 583
- Vogt S. S., Penrod G. D., Hatzes A. P., 1987, *ApJ*, 321, 496
- Wilson R. E., Devinney E. J., 1971, *ApJ*, 166, 605

⁵ <https://www.khronos.org/opencl/>