Assembly-Free Large-Scale Modal Analysis on the GPU

Praveen Yadav, Krishnan Suresh

suresh@engr.wisc.edu

Department of Mechanical Engineering,

UW-Madison, Madison, Wisconsin 53706, USA

Abstract

Popular eigen-solvers such as block-Lanczos require repeated inversion of an eigen-matrix. This is a bottleneck in large-scale modal problems with millions of degrees of freedom. On the other hand, the classic Rayleigh-Ritz conjugate gradient method only requires a matrix-vector multiplication, and is therefore potentially scalable to such problems. However, as is well-known, the Rayleigh-Ritz has serious numerical deficiencies, and has largely been abandoned by the finite element community.

In this paper, we address these deficiencies through subspace augmentation, and consider a subspace augmented Rayleigh-Ritz conjugate gradient method (SaRCG). SaRCG is numerically stable and does not entail explicit inversion. As a specific application, we consider the modal analysis of geometrically complex structures discretized via non-conforming voxels. The resulting large-scale eigen-problems are then solved via SaRCG. The voxelization structure is also exploited to render the underlying matrix-vector multiplication assembly-free. The implementation of SaRCG on multi-core CPUs, and graphics-programmable-units (GPUs) is discussed, followed by numerical experiments and case-studies.

1. INTRODUCTION

Consider the modal analysis of a thin geometrically complex structure illustrated in Figure 1. The first step in such an analysis is the construction of a suitable finite-element mesh.

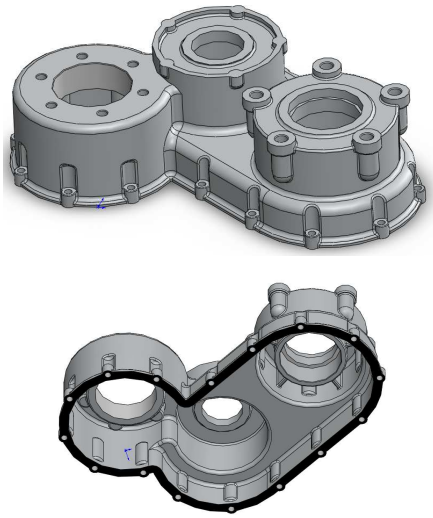


Figure 1: A thin gear-housing whose eigen-spectrum is desired.

Thin geometrically complex structures require a small element size … resulting in a large scale finite-element problem. For example, for the above structures, after numerous failed attempts, we were able to create the tetrahedral mesh illustrated in Figure 2, with over 750,000 degrees of freedom.

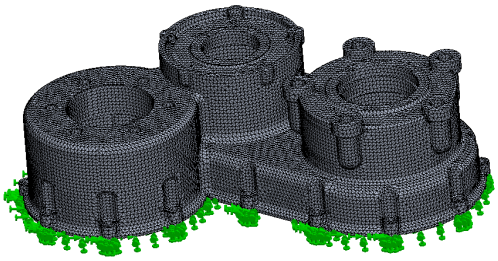


Figure 2: Complex thin-structures require a fine-mesh.

This leads to the eigen-problem:

$$Kx = \lambda Mx \tag{1.1}$$

where K & M are the stiffness and mass matrices associated with the finite-element discretization. The above problem has been well-studied, and numerous implementations exist [1–3]. When K & M are large, practical limitations arise. Specifically, popular eigen-solvers such as block-Lanczos require explicit and repeated inversion of a large-size eigen-matrix (see Section 2), and the memory requirements and computational time grows quadratically in such methods.

On the other hand, the classic Rayleigh-Ritz conjugate gradient method for solving Equation (1.1) only requires an implementation of sparse matrix-vector multiplication. The memory requirements are low, and the computational complexity depends largely on the efficiency of the matrix-vector multiplication. However, as numerical experiments indicate, the Rayleigh-Ritz method lacks robustness, and can exhibit poor convergence; consequently, it has largely been abandoned by the finite element community.

In this paper, we revisit this method, and address the limitations through subspace augmentation. The resulting subspace augmented Rayleigh-Ritz conjugate gradient (SaRCG) method is highly robust, exhibits fast convergence, and is easily parallelizable. An assembly-free implementation of SaRCG on multi-core and GPU architectures is discussed.

2. LITERATURE REVIEW

In this Section, we briefly review popular strategies for solving the generalized eigen-value problem:

$$Kx = \lambda Mx \tag{2.1}$$

For the remainder of the paper, we assume that K and M are sparse, symmetric, real, and positive definite.

One of the most popular methods today to solve Equation (2.1) is the block-form of the shift-and-invert Lanczos algorithm, also referred as the block-Lanczos method [2], [4], [5]. The block-Lanczos method requires repeated 'inversion' of an eigen-matrix $K - \sigma M$, where the parameter σ is determined during iterations of the method.

Since K and M are large, and the parameter σ varies during the iteration, explicit LU factorization of $K - \sigma M$ is not desirable. An alternate strategy is to factorize this matrix through preconditioned iterative solvers. However, as emphasized in [2], [6], early termination of iterative solvers can lead to a significant loss of accuracy in block-Lanczos, requiring careful attention to termination.

In [6], the classic block-Lanczos algorithm was instead modified to eliminate the need for an explicit decomposition. Specifically, the 'inversion' was carried out in an approximate sense over a Krylov sub-space. The method discussed in this paper borrows this concept, but in a different context.

In [2], the authors rely on Algebraic Multigrid as a pre-conditioner for factorizing the shifted matrix. In addition, the authors also implement and compare a variety of alternate algorithms including 'locally optimal block preconditioned conjugate gradient', 'Davidson-Jacobi', etc. More importantly, the authors demonstrate that, for large-scale eigen-value problems, such alternate algorithms can be competitive. This is a key motivation for the present paper.

One such alternate algorithm is the Rayleigh-Ritz conjugate gradient (RCG) where the eigen-mode problem is posed and solved as a minimization problem (see next Section). Since the RCG method can be slow to converge, the authors in [7] implement a parallel version of RCG with factorized sparse approximate inverse as a pre-conditioner. Further, in [8], a locally-optimal variation of RCG was proposed. The essential idea is to expand the search-space during each step of CG in a locally-optimal sense. We do not implement this variation here, but the proposed method can be modified to include local-optimality.

Besides block-Lanczos and RCG, there are dozens of methods including Jacobi-Davidson [9] and inverse-iteration [10] that do not offer significant advantages over RCG or block-Lanczos, and are therefore not pursued here.

3. Rayleigh-Ritz Conjugate Gradient

As stated earlier, the Rayleigh-Ritz conjugate gradient (RCG) algorithm [11], [12] only requires an efficient implementation of sparse matrix-vector multiplication. It therefore exhibits numerous advantages including simplicity, low memory requirements, and significant scope for parallelism.

3.1 Rayleigh Quotient Concept

A key concept behind RCG is the Rayleigh quotient of an arbitrary vector x:

$$\varphi(x) = \frac{x^T Kx}{x^T Mx} \tag{3.1}$$

If the vector x happens to be an eigen-vector of (K, M), then the Rayleigh quotient is the corresponding eigen-value. Thus, by minimizing the Rayleigh quotient, one can compute the lowest eigen-pair, i.e., the eigen-value problem can be posed as a minimization problem:

$$\underset{x}{Min} \frac{x^T Kx}{x^T Mx} \tag{3.2}$$

3.2 Computing the Lowest Mode

Thus, one can deploy, for example, the nonlinear conjugate gradient method [13] to find the lowest eigen-mode. Towards this end, observe that the gradient of Equation (3.2) is given by:

$$g(x) = 2\left(\frac{Kx - \varphi(x)Mx}{\left\|x\right\|_M^2}\right) \tag{3.3}$$

where:

$$\left\|x\right\|_M = \sqrt{x^T Mx} \tag{3.4}$$

Exploiting the classic conjugate gradient algorithm [13], we have the Rayleigh-Ritz conjugate gradient (RCG; Version 1):

Rayleigh-Ritz Conjugate Gradient (Version 1):

1. Initialize $x^{(1)} \neq 0$ such that $\left\|x^{(1)}\right\|_M = 1$
2. Set $p^{(0)} = 0$, $\beta^{(1)} = 1$ and $k = 1$
3. Compute $\varphi(x^{(k)})$ via Equation (3.1), and the gradient $g^{(k)}$ via Equation (3.3)

4. The conjugate search direction is given by:

   $p^{(k)} = -g^{(k)} + \beta^{(k)} p^{(k-1)}$

5. Find the step length $\delta^{(k)}$ as described in [14]

6. Let $y^{(k+1)} = x^{(k)} + \delta^{(k)} p^{(k)}$ and

   $x^{(k+1)} = y^{(k+1)} / \left\| y^{(k+1)} \right\|_M$

7. If $\left\| g^{(k)} \right\| \le \varepsilon$, terminate; else, increment k, and

   go to step 3

Observe that the RCG algorithm only requires matrix-vector multiplications: Kv and Mv, making it a simple algorithm to implement.

3.3 Computing Multiple Modes

To compute higher modes, observe that that if (λ_1, x_1) and (λ_2, x_2) are two distinct modes, and if $\lambda_1 \neq \lambda_2$, then they must satisfy M-orthogonality:

$$x_1^T M x_2 = 0 \tag{3.5}$$

Further, if $\lambda_1 = \lambda_2$, one can always find a pair of eigen-modes that satisfy the above equation. Thus, to find the second eigen-mode, we pose a *constrained* minimization problem:

$$\begin{aligned} &\underset{x}{Min} \frac{x^T K x}{x^T M x} \\ &s.t. \;\; x^T M x_1 = 0 \end{aligned} \tag{3.6}$$

Given an arbitrary vector x, one can enforce M-orthogonality via:

$$x = x - \left(x^T M x_1 \right) x_1 \tag{3.7}$$

To ensure M-orthogonality during the iteration process, it is necessary and sufficient if: (1) the initial random vector is M-orthogonal to x_1, and (2) the search directions $p^{(k)}$ are M-orthogonal to x_1.

Thus, suppose the first (m-1)-modes have been computed:

$$X = \{x_1, x_2, ..., x_{m-1}\} \tag{3.8}$$

To compute the next mode, one must solve the constrained minimization problem:

$$\begin{aligned} &\underset{x}{Min} \frac{x^T K x}{x^T M x} \\ &s.t. \;\; x^T M X = 0 \end{aligned} \tag{3.9}$$

where M-orthogonality is enforced via:

$$x = x - \sum_{i=1}^{m-1} \left(x^T M x_i \right) x_i \tag{3.10}$$

This leads to the following RCG algorithm (Version 2) for computing the mth pair (λ_m, x_m) [11].

Rayleigh-Ritz Conjugate Gradient (Version 2):

1. Suppose m-1 eigen-modes $X = \{x_1, x_2, ..., x_{m-1}\}$ have been computed.
2. Initialize $x^{(1)} \neq 0$ such that $\left\| x^{(1)} \right\|_M = 1$ and $x_1^T M X = 0$
3. Set $p^{(0)} = 0$, $\beta^{(1)} = 1$ and $k = 1$
4. Compute $\varphi(x^{(k)})$ via Equation (3.1), and the gradient $g^{(k)}$ via Equation (3.3)
5. Let $\tilde{p}^{(k)} = -g^{(k)} + \beta^{(k)} p^{(k-1)}$ (preliminary direction)
6. Construct an M-orthogonal direction $p^{(k)}$ via Equation (3.10)
7. Find the step length $\delta^{(k)}$ as described in [14]
8. Let $y^{(k+1)} = x^{(k)} + \delta^{(k)} p^{(k)}$ and $x^{(k+1)} = y^{(k+1)} / \left\| y^{(k+1)} \right\|_M$
9. If $\left\| g^{(k)} \right\| \le \varepsilon$, terminate; else, increment k, and go to step 4.

3.4 Subspace Augmentation

There are three fundamental limitations of the above RCG algorithm; these are confirmed later through numerical experiments. To the best of our knowledge, these limitations have not been identified/ emphasized in the literature:

1. **Missing modes**: As we sweep through the eigen-spectrum, one or more eigen-modes may go undetected, especially in the case of 'near-by' eigen-values. This can be attributed to the fact that RCG attempts to find local minima, and presence of multiple minima that are close-by can be detrimental to the algorithm.
2. **Erroneous Results**: A large value for ε (in step-9 of RCG, version 2) can leads to erroneous results; typically $\varepsilon \sim 10^{-10}$ is essential.
3. **Slow convergence**: On the other hand, $\varepsilon \sim 10^{-10}$ leads to thousands of iterations (for each eigen-mode), especially for ill-conditioned matrices.

Fortunately, all three problems can be addressed by relying on subspace projection methods. Such projection methods are fairly common in modern eigen-solvers [3]. In particular, the method proposed

here is similar to the generic method of subspace projection discussed in [2].

Specifically, let us assume that, at the end of k iterations, we have a set of approximate eigen-vectors $\tilde{x}_i^{(k)}(i=1,2,...,m)$ computed via early termination of RCG (say $\varepsilon \sim 0.01$). A sub-space is then defined via:

$$S^{(k)} = \{\tilde{x}_1^{(k)}, \tilde{x}_2^{(k)}, ..., \tilde{x}_m^{(k)}\} \tag{3.11}$$

Next, a reduced stiffness and mass matrices are constructed over this sub-subspace as follows:

$$\begin{aligned} \bar{K}^{(k)} &= S^{(k)T} K S^{(k)} \\ \bar{M}^{(k)} &= S^{(k)T} M S^{(k)} \end{aligned} \tag{3.12}$$

Observe that computing these matrices only requires sparse matrix-vector multiplications. The two matrices are then used to solve a smaller eigen-value problem (exactly) via:

$$\bar{K}^{(k)} V^{(k)} = \bar{M}^{(k)} V^{(k)} \Lambda^{(k)} \tag{3.13}$$

Finally, a sharpened set of eigen-vectors are recovered via:

$$X^{(k+1)} = S^{(k)} V^{(k)} \tag{3.14}$$

where:

$$X^{(k+1)} = \{x_1^{(k+1)}, x_2^{(k+1)}, ..., x_m^{(k+1)}\} \tag{3.15}$$

One can now restart the RCG method with the eigen-vectors in Equation (3.15) as the starting vectors. This leads to the following subspace augmented Rayleigh-Ritz conjugate gradient algorithm (SaRCG), summarized below.

Subspace Augmented Rayleigh-Ritz Conjugate Gradient (SaRCG):

1. Initialize $X^{(1)} = \{x_1^{(1)}, x_2^{(1)}, ..., x_m^{(1)}\}$ (typically random vectors). Set $k = 1$
2. Compute $\tilde{x}_i^{(k)}$ with $x_i^{(k)}$ as the starting vectors until $\left\| g^{(k)} \right\| \le \varepsilon$ ($\varepsilon \sim 0.01$), for $i = 1,2,...,m$, via RCG (version 2).
3. Construct the reduced matrices via Equation (3.12), where $S^{(k)}$ is defined in Equation (3.11), and solve Equation (3.13) to find $\Lambda^{(k)}$ and $V^{(k)}$.
4. If convergence in $\Lambda^{(k)}$ has not been achieved, construct the updated eigen-vectors $X^{(k+1)}$ via Equation (3.14), increment k and go to step 2.

This simple extension of the RCG algorithm dramatically improves its robustness and convergence. The additional computational cost of constructing and solving the reduced eigen-value problem is negligible. In other words, the primary cost in SaRCG, is the sparse matrix-vector multiplication (SpMV). Therefore, an efficient implementation of SpMV is critical.

3.5 Voxelization

The proposed SaRCG method is applicable to any finite-element discretization, including the tetrahedral discretization of Figure 2. However, in this paper, we consider a simple discretization, where the geometry is approximated via uniform cubes or 'voxels'; the voxel-approach has gained significant popularity recently [15]. Since the voxelization need not conform to the geometry, it is robust, fast and relatively insensitive to the geometric complexity. Further, unlike in stress analysis, where a conforming mesh is essential, in eigen-mode analysis, a non-conforming voxelization is often sufficient, especially during the early stage of design. This is confirmed through numerical experiments later in the paper.

The voxelization of the geometry in Figure 2 is illustrated in Figure 3; it has over a million degrees of freedom. Fortunately, even such a large-size problem is easily handled via the proposed method.

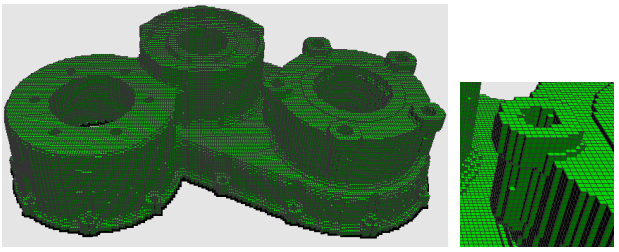


Figure 3: Brute-force voxelization of the structure.

The voxelization of a triangulated CAD model, also referred to as 3-D scan conversion, is straight-forward, and is discussed, for example, see [16]. Voxels that are partially inside the geometry are assigned a density value depending on the number of corner nodes that lie inside the geometry. As the numerical experiments indicate, the overall computational cost for the voxelization is small compared to the cost of computing the eigen-modes.

3.6 Shape Functions

Given a voxelization, one can choose a variety of hexahedral finite element shape functions. The simplest are tri-linear shape functions as described in [17], where each node-based shape function is of the form:

$$N_i = 0.125(1+\xi_i\xi)(1+\eta_i\eta)(1+\zeta_i\zeta); i = 1...8 \tag{3.16}$$

However, the resulting 8-noded elements are 'stiff', and convergence is slow especially for higher order modes, as illustrated through numerical experiments. One could use 20-node or 27-node elements, but these increase the memory requirements significantly.

In the context of static problems the authors of [18] instead propose the use of three additional bubble-functions of the form:

$$M_1(\xi,\eta,\zeta) = (1-\xi^2)$$
$$M_2(\xi,\eta,\zeta) = (1-\eta^2) \quad (3.17)$$
$$M_3(\xi,\eta,\zeta) = (1-\zeta^2)$$

This resulting stiffness matrix will be form:

$$\bar{K}_e = \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} \quad (3.18)$$

where

$$K_{11} = \int \nabla N^T D \nabla N d\Omega$$
$$K_{12} = \int \nabla N^T D \nabla M d\Omega \quad (3.19)$$
$$K_{21} = \int \nabla M^T D \nabla N d\Omega$$
$$K_{22} = \int \nabla M^T D \nabla M d\Omega$$

Fortunately, one can condense out the bubble degrees of freedom, resulting again in a reduced 24 degrees of freedom element stiffness matrix [18]:

$$K_e = K_{11} - K_{12}(K_{22} \setminus K_{21}) \quad (3.20)$$

While the authors in [18] demonstrate the merit of this formulation for static problems, we have adopted these bubble functions for eigen-problems. Numerical experiments indicate bubble functions are highly effective in eigen-problems as well.

3.7 Assembly-Free SpMV

The primary computational cost of sparse matrix-vector multiplication (SpMV), in today's computational architecture, is memory-access, i.e., floating-point operations are essentially free [2]. In SpMV, there are two types of memory access: (1) accessing elements of $x_i^{(k)}$, and (2) accessing elements of K and M.

There is very little one can do about the former, except perhaps careful numbering of the mesh nodes. However, the cost of accessing elements of K and M can be dramatically reduced through assembly-free methods as discussed below.

Since the geometry is discretized via uniform voxels, all elements are geometrically identical. Therefore all element matrices are identical (barring a 'density' assigned to partial voxels). Thus, one need not assemble or store the global K and M matrices; it is sufficient if a single matrix-pair K_e and M_e is stored.

Consequently, a matrix-free implementation of Kv, for example, may be implemented as follows [19]:

$$Kx = \left(\sum_e K_e\right) x = \sum_e \left(K_e x_e\right) \quad (3.21)$$

In other words, instead of assembling the K matrix, and then carrying out Kx, we first multiply $K_e x_e$ and then assemble the result.

3.8 CPU and GPU Implementations

In the CPU implementation of the assembly-free SpMV, parallelization was attained through OpenMP commands (www.openmp.org).

In the GPU implementation using CUDA [20] on NVidia cards, SpMV was implemented by assigning each node of the voxel-grid to a scalar processor. Associated with each voxel node, there are at most 8 voxel elements, and '27' neighboring nodes. When a block of threads are launched, all threads share identical element matrices, and therefore a single memory fetch of the stiffness and mass element matrix per block is sufficient.

The three degrees of freedom associated with the 27 nodes are fetched in parallel. In general, during this fetch, it is hard to enforce coalesced memory access for arbitrary voxel-meshes. For more recent GPU cards, we have observed that this is not a serious issue. Finally, many of the low-level operations such as vector-product were implemented using CUBLAS library.

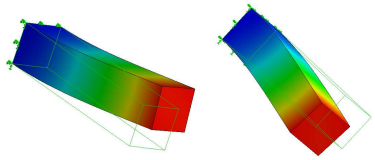
4. NUMERICAL EXPERIMENTS

In this Section, we present results from numerical experiments based on the RCG and SaRCG algorithms. All experiments were conducted on a Windows-7 64-bit machine with the following hardware:

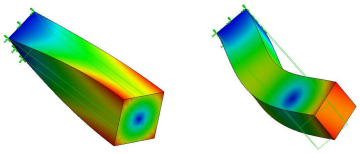
- Intel I7 960 CPU quad-core running at 3.2GHz with 6 GB of memory; parallelization of CPU code was implemented through OpenMp commands.
- The graphics programmable unit (GPU) is an NVidia GeForce GTX 480 (480 cores) with 1.5 GB.
- Both the CPU and GPU were configured to run in double-precision.

4.1 Importance of Bubble Functions

In this experiment, we illustrate the value added by the use of bubble functions discussed earlier. Specifically, a short cantilevered beam of dimension 1 x 1 x 5, fixed on end is used in this experiment. In Figure 4, the first 4 eigen-modes are illustrated; these were computed using a quadratic tetrahedral mesh with 45,000 DOF, in SolidWorks [21].



(1) 3221 Hz (2) 3221 Hz



(3) 14728 Hz (4) 17412 Hz

Figure 4: First four modes computed via SolidWorks.

Next using SaRCG, the first eigen-mode was computed for different voxel sizes, with-and-without bubble function. As one can observe, the use of bubble functions dramatically improves the convergence at no additional cost. Similar improvements were observed for higher order modes, and for other case-studies. Therefore, for the remainder of the experiments, bubble functions are employed.

Table 1: First eigen-value of short cantilevered beam, with-and-without bubble functions.

#DOF	Without Bubble (Hz)	%error	With Bubble (Hz)	%error
1650	3278	1.8	3230	0.027
10000	3236	0.20	3224	0.009
31000	3228	0.05	3222	0.003
48000	3226	0.016	3221	0.000

4.2 Deficiency of RCG

Next, we illustrate the deficiency of RCG. Using the same cantilevered beam example, we computed the first four eigen-modes via RCG for different tolerances values ε (see Version 2 of RCG algorithm) for a fixed voxel size (55,000 DOF). As one can observe in Table 2, even with tight tolerances, RCG can miss some of the critical modes or compute them out of order … the classic RCG method is not robust.

Table 2: First 4 eigen-value of beam via RCG.

	Mode-1	Mode-2	Mode-3	Mode-4
$\varepsilon = 10^{-8}$	3260	17423	3566	25611
$\varepsilon = 10^{-9}$	3223	3230	17455	25610
$\varepsilon = 10^{-10}$	3222	3222	17455	22846
$\varepsilon = 10^{-11}$	3222	3222	17455	14985

On the other hand, SaRCG converges to the correct set of eigen-modes, even with a coarse tolerance of $\varepsilon = 10^{-4}$, as summarized in Table 3.

Table 3: First 4 eigen-values of beam via SaRCG.

	Mode-1	Mode-2	Mode-3	Mode-4
$\varepsilon = 10^{-4}$	3221	3221	14851	17455
$\varepsilon = 10^{-11}$	3221	3221	14851	17455

4.2 Beam: Conforming vs. Non-Conforming

A premise of this paper is that, for approximate eigen-mode analysis, a non-conforming mesh may be sufficient. We illustrate this through the following experiment. The beam from the previous example was rotated about its longest axis by an arbitrary angle of 11 degrees, and the resulting geometry was discretized via a non-conforming set of voxels (see Figure 5). Then the first four eigen-modes were computed for increasing density of the mesh, using SaRCG with bubble-mesh, and a tolerance of $\varepsilon = 10^{-4}$.

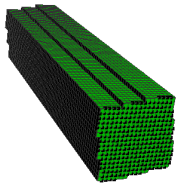


Figure 5: Non-conforming voxelization of a beam.

Table 4 summarizes the results of this experiment. Clearly, non-conforming meshes are never as accurate as conforming meshes (for a fixed grid-size). However, the inaccuracy can be reduced through brute-force computation.

Table 4: First eigen-value for cylindrical beam cantilever using SaCG with tolerance of 10^{-2}.

#DOF	GPU Time (sec)	Mode-1	Mode-2	Mode-3	Mode-4
55000	9.2	3201	3226	15138	17413
110000	19.2	3202	3224	14991	17427
200000	26.9	3215	3216	14892	17420
300000	44.9	3219	3223	14787	17435

4.3 Crank-Rod: A Practical Example

A more realistic example is that of a crank-rod illustrated in Figure 6 that is clamped on the inner-surface of the smaller cylindrical hole. The first eigen-mode was computed using SolidWorks with default mesh parameters. The resulting conforming (second-order tetrahedral) mesh contains approximately 90,000 degrees of freedom. The first eigen-mode was computed to be 529.6 Hz; the total computational time including meshing and solver time was approximately 9 seconds. Refinement of the mesh did not change the first eigen-mode significantly in SolidWorks.

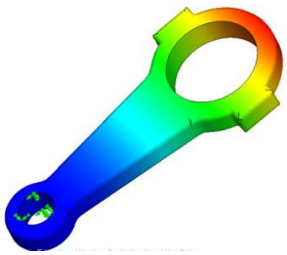


Figure 6: First eigen-mode of a connecting-rod at 529.6 Hz

In Table 5, the computed eigen-value (via SaRCG) as the mesh is refined is summarized. As one can observe, the eigen-value converges to the 'correct' value, and the computational time, is comparable to that of SolidWorks, especially when a GPU such as the GTX 480 is used.

Table 5: First eigen-value for crank-rod.

DOF	Mode-1	Voxelization time (CPU)	Solution time (GPU)	Solution time (CPU)
9,000	515 Hz	0.07 secs	1.6 secs	2.7 secs
25,000	560 Hz	0.12 secs	2.1 secs	3.7 secs
80,000	542 Hz	0.4 secs	4 secs	22 secs
136,000	538 Hz	0.6 secs	7 secs	58 secs
250,000	533 Hz	1.2 secs	16 secs	151 secs

4.4 Knuckle: Accuracy of First Few Eigen-Modes

In the next example, we compare the accuracy of SaRCG method for the first five eigen-modes for the 'knuckle' problem illustrated in Figure 6a, where the two horizontal holes are clamped. Also illustrated in Figure 6b and Figure 6c are the confirming and non-conforming meshes respectively, with approximately 150,000 and 250,000 degrees of freedom, respectively.

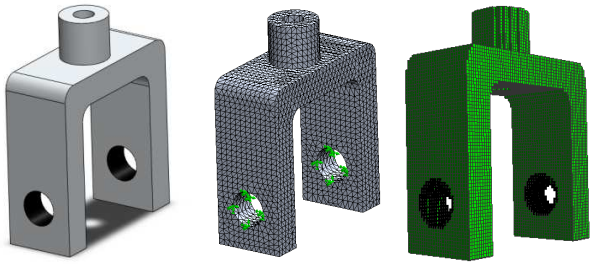
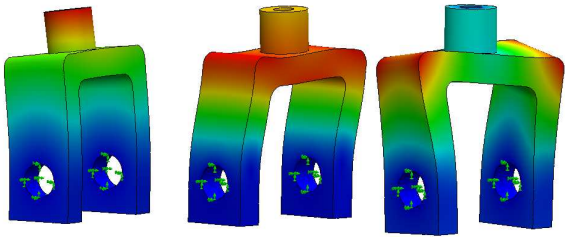
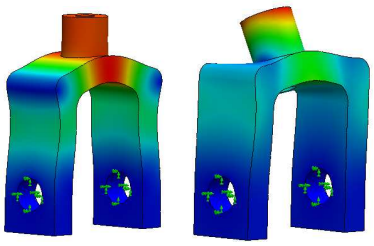


Figure 7: (a) A knuckle component, (b) conforming mesh, and (c) voxel-mesh

The first five eigen-modes as computed using SolidWorks, are illustrated in Figure 8; the total computational time was approximately 15 seconds.



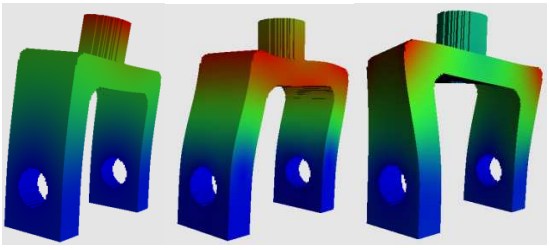
(1) 2341 Hz (2) 2484 Hz (3) 4221 Hz



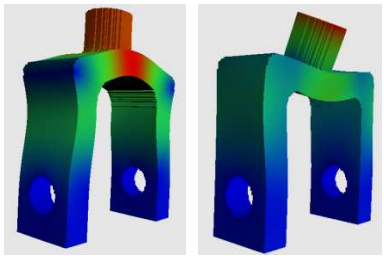
(4) 5856 Hz (5) 7595 Hz

Figure 8: The first 5 modes computed via SolidWorks.

The corresponding five eigen-modes computed via the proposed method are illustrated in Figure 9. The eigen-values are within 1~2% accuracy, and the computational time was approximately 35 seconds, on the GPU, and 1.5 minutes on the CPU. For the next 20 modes, convergence to within 3~5% was also observed.



(1) 2393 Hz (2) 2445 Hz (3) 4261 Hz



(4) 5724 Hz (5) 7534 Hz

Figure 9: The first 5 modes computed via proposed method.

4.5 Gear Housing: Robustness

The primary advantages of the proposed method are its robustness, simplicity and ability to handle geometrically complex structures. To illustrate, consider the gear housing illustrated earlier in Figure 1. The first eigen-mode of the structure computed via the proposed method is illustrated in Figure 10.

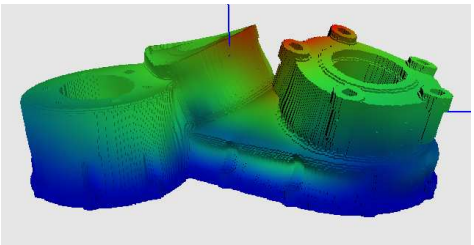


Figure 10: The first mode of the structure in Figure 1.

The convergence of the first eigen-value with voxel-size is summarized in Table 6; it demonstrates that one can rapidly estimate the eigen-values in a fully automated fashion. This is particularly important during the early stages of design.

Table 6: First eigen-value for gear-housing.

DOF	Mode-1	Solution time (GPU)	Solution time (CPU)
150,000	70	7.1 secs	62 secs
300,000	76	18 secs	2.1 mins
425,000	74.2	43 secs	3.7 mins
2,000,000	74.6	191 secs	23 mins

5. CONCLUSION

The main contribution of this paper is an assembly-free implementation of Subspace augmented Rayleigh-Ritz Conjugate Gradient (SaRCG), for computing the eigen-modes of elastic structures. The proposed method is simple, robust and 'inverse-free'. Since the method only requires an implementation of a sparse matrix vector multiplication (SpMV), it is highly parallelizable, and can be easily implemented on modern multi-core architectures. Although the method was demonstrated using a non-conforming structured (voxel) mesh, it is equally applicable to classic conforming meshes.

6. REFERENCES

[1] V. Hernandez, J. E. Roman, A. Tomas, and V. Vidal, "A survey of software for sparse eigenvalue problems.," *SLEPc Technical Report STR-6. Universidad Politecnica de Valencia, Valencia, Spain. http://www.grycap.upv.es/slepc.*, 2009.

[2] P. Arbenz, U. L. Hetmaniuk, R. B. Lehoucq, and R. S. Tuminaro, "A Comparison of Eigensolvers for Large-scale 3D Modal Analysis using AMG-Preconditioned Iterative Methods," *International Journal for Numerical Methods in Engineering*, vol. 64, no. 2, pp. 204–236, 2005.

[3] Y. Saad, *Numerical methods for large eigenvalue problems*, 2nd ed. Manchester University Press, 2011.

[4] R. G. Grimes, J. G. Lewis, and H. D. Simon, "A Shifted Block Lanczos Algorithm for Solving Sparse Symmetric Generalized Eigenproblems," *SIAM Journal on Matrix Analysis and Applications*, vol. 15, no. 1, p. 228, 1994.

[5] D. C. Sorensen, "Numerical methods for large eigenvalue problems," *ACTA NUMERICA*, vol. 11, pp. 519–584, 2002.

[6] G. H. Golub and Q. Ye, "An Inverse Free Preconditioned Krylov Subspace Method for Symmetric Generalized Eigenvalue Problems," *SIAM Journal on Scientific Computing*, vol. 24, no. 1, pp. 312–334, 2002.

[7] L. Bergamaschi, Á. Martínez, and G. Pini, "Parallel preconditioned conjugate gradient optimization of the Rayleigh quotient for the solution of sparse eigenproblems," *Applied mathematics and computation*, vol. 175, no. 2, p. 1964, 2006.

[8] A. V. Knyazev, "Toward the Optimal Preconditioned Eigensolver: Locally Optimal Block Preconditioned Conjugate Gradient Method," *SIAM Journal on Scientific Computing*, vol. 23, no. 2, pp. 517–541, 2001.

[9] G. L. G. Sleijpen and H. A. Van der Vorst, "A Jacobi-Davidson Iteration Method for Linear Eigenvalue Problems," *SIAM Journal on Matrix Analysis and Applications*, vol. 17, no. 2, pp. 401–425, 1996.

[10] I. C. F. Ipsen, "Computing an Eigenvector with Inverse Iteration," *SIAM REVIEW*, vol. 39, no. 2, pp. 254–291, 1997.

[11] H.-J. Jang, "Preconditioned Conjugate Gradient Method for Large Generalized Eigenproblems," *Trends in Mathematics Information Center for Mathematical Sciences*, vol. 4, no. 2, pp. 103–109, 2001.

[12] Y. T. Feng and D. R. J. Owen, "Conjugate Gradient Methods for Solving the Smallest Eigenpair of Large Symmetric Eigenvalue Problems," *International Journal for Numerical Methods in Engineering*, vol. 39, no. 13, pp. 2209–2230, 1996.

[13] J., Wright, S. Nocedal, *Numerical optimization*. New York: Springer Science + Business Media, 2006.

[14] H. Yang, "Conjugate Gradient Methods for the Rayleigh Quotient Minimization of Generalized Eigenvalue Problems," *Computing -Wein-*, vol. 51, no. 1, pp. 79–94, 1993.

[15] A. Duster, J. Parvizian, Z. Yang, and E. Rank, "The Finite Cell Method for 3D problems of solid mechanics.," *Computer Methods in Applied Mechanics and Engineering*, vol. 197, pp. 3768–3782, 2008.

[16] E. A. Karabassi, G. Papaioannou, and T. Theoharis, "A Fast Depth-Buffer-Based

Voxelization Algorithm," *Journal of Graphics Tools*, vol. 4, no. 4, 1999.

[17] O. C. Zienkiewicz, *The Finite Element Method for Solid and Structural Mechanics*. Elsevier, 2005.

[18] H. H. Taiebat and J. P. Carter, "Three-Dimensional Non-Conforming Elements," Centre for Geotechnical Research, The University of Sydney, Sydney, R808, 2001.

[19] C. E. Augarde, A. Ramage, and J. Staudacher, "An element-based displacement preconditioner for linear elasticity problems," *Computers and Structures*, vol. 84, no. 31–32, pp. 2306–2315, 2006.

[20] NVIDIA Corporation, *NVIDIA CUDA: Compute Unified Device Architecture, Programming Guide*. Santa Clara.: , 2008.

[21] SolidWorks, *SolidWorks; www.solidworks.com*. 2005.