# **Lisp and the Implementation of Primitive Drawing Commands**

## **1. Introduction to Lisp and its Significance**

Lisp stands as a foundational pillar in the landscape of programming languages, distinguished by its age and its profound influence on the evolution of computing. Second only to Fortran in the lineage of high-level languages, Lisp's origins trace back to the late 1950s.1 This remarkable longevity suggests an inherent strength in its fundamental design, enabling it to adapt and remain relevant across decades of technological advancement. Its development was spearheaded by John McCarthy at the Massachusetts Institute of Technology (MIT), driven by the need for a language adept at Artificial Intelligence (AI) research.1 This initial focus on AI has indelibly shaped Lisp's characteristics and its perception, often associating it with symbolic computation and knowledge representation rather than traditional, general-purpose programming paradigms.

Despite this strong association with AI, Lisp's impact extends far beyond its original domain. It is credited with pioneering fundamental concepts in computer science that have subsequently influenced the design of nearly every modern programming language.10 Ideas such as recursion, automatic memory management through garbage collection, dynamic typing, and the revolutionary concept of treating code as data were either first introduced or significantly popularized by Lisp. These innovations have had a lasting impact on how we approach software development today. Furthermore, while perhaps less widely known than its contributions to AI, Lisp has also played a significant role in the history and development of computer graphics.1 This report will delve into the core concepts of Lisp, its historical involvement in computer graphics, and specifically examine the implementation of primitive drawing commands within the language and its associated libraries.

## **2. Core Concepts of the Lisp Programming Language**

The syntax of Lisp is fundamentally based on S-expressions (Symbolic Expressions), a remarkably simple yet powerful way of structuring data and code.1 The defining characteristic of S-expressions is the pervasive use of parentheses to delineate lists and function calls. This uniform syntax, where both data and program instructions share the same list-based structure, is a cornerstone of Lisp's flexibility and enables a powerful feature known as metaprogramming, where programs can manipulate other programs as data.13 This capability is central to Lisp's extensibility and its ability to adapt to diverse problem domains.

At its core, Lisp revolves around three fundamental data structures: atoms, lists, and symbols.1 Atoms represent basic data elements like numbers and strings, while lists are ordered collections of items, which can themselves be atoms or other lists, allowing for the creation of arbitrarily complex hierarchical structures. Symbols are unique named objects that serve as identifiers for variables, functions, and other program entities. The language's emphasis on lists as a primary data structure, reflected in its very name derived from "List Processor" 10, makes it particularly well-suited for symbolic manipulation and processing hierarchical data, which proves invaluable in representing graphical structures and relationships.

Lisp is primarily considered a functional programming language, where computation is expressed as the evaluation of functions.1 This paradigm encourages the development of programs through the composition of functions, promoting principles like immutability (data structures are not modified after creation) and the use of higher-order functions (functions that can take other functions as arguments or return them as results). These functional programming tenets can lead to more concise, elegant, and maintainable code, offering potential benefits in the development of graphics algorithms and applications.

One of Lisp's most distinctive and potent features is its macro system.1 Macros provide a mechanism for metaprogramming, allowing programmers to define new syntactic constructs and effectively extend the language's syntax to suit specific needs.1 This capability enables the creation of domain-specific languages tailored to particular problem areas, such as computer graphics, where custom syntax for defining drawing operations or geometric transformations can significantly enhance expressiveness and reduce boilerplate code.

Over its long history, Lisp has evolved into a family of languages known as dialects. Among these, two have achieved widespread recognition and continue to be actively used: Common Lisp and Scheme.1 While both share the fundamental principles of Lisp, they have diverged in their development, resulting in variations in their features, standard libraries, and community focus. Understanding these differences is crucial, especially when exploring graphics libraries and historical usage, as these might be specific to one dialect or the other.

## **3. A Historical Perspective: Lisp and Computer Graphics**

Lisp's involvement in computer graphics dates back to the early days of the field, with its flexibility and symbolic processing capabilities proving attractive for pioneering research and the development of innovative visual systems.1 The ability to easily manipulate symbolic representations of graphical objects and algorithms made Lisp a natural choice for early graphics experiments, where rapid prototyping and the exploration of novel visual representations were paramount. For instance, the development of LOGLISP 20, which uniquely amalgamated Lisp and logic programming, suggests early explorations into the intersection of these paradigms within the realm of graphics. Similarly, the mention of design and graphic description systems built upon Lisp 21 further underscores its early adoption in this domain.

Lisp played a significant role in the evolution of interactive graphics systems, particularly through the development of Lisp machines.11 These specialized computers, engineered to efficiently run Lisp as their primary language, pioneered many technologies that are now commonplace in graphical computing. The emergence of windowing systems, computer mice, and high-resolution bit-mapped raster graphics on Lisp machines during the 1980s demonstrates Lisp's early and substantial contribution to the foundations of modern graphical user interfaces.

A notable example of Lisp's practical application in graphics is its adoption as an extension language in commercial Computer-Aided Design (CAD) software, most prominently with AutoLISP in AutoCAD.11 Since its integration in 1985, AutoLISP has become widely used by AutoCAD professionals to automate repetitive tasks, customize workflows, and extend the software's functionality. This widespread use in a professional CAD environment highlights the practical applicability of Lisp for enhancing and automating graphical design processes.

Beyond these broader trends, specific examples further illustrate Lisp's historical involvement in computer graphics. The development of a piano-roll notation program in Lisp 37 provides a concrete instance of the language being used for the visual representation of musical data. This example showcases Lisp's capacity to handle complex symbolic structures and translate them into graphical output. The fact that animation and graphics are listed as application areas where Lisp has been successfully applied 10 reinforces its versatility beyond purely symbolic AI tasks.

## **4. Implementing Primitive Drawing Commands in Lisp**

The implementation of basic drawing primitives in Lisp involves fundamental concepts of representing and manipulating graphical data. At a low level, drawing on a digital display ultimately boils down to controlling the color of individual pixels. However, higher-level primitives like points, lines, circles, and rectangles abstract away this pixel-level manipulation, providing more convenient building blocks for creating graphics.

In Lisp, graphical data can be represented in various ways, leveraging the language's flexible data structures. For instance, a point can be represented as a list of two numbers corresponding to its x and y coordinates, such as (10 20). A line segment could be represented by a list containing two points, like ((10 20) (30 40)). More complex shapes like circles and rectangles might be represented using lists containing their defining parameters, such as (circle (50 50) 25) for a circle with center at (50, 50) and radius 25, or (rectangle (10 10) 40 30) for a rectangle with top-left corner at (10, 10), width 40, and height 30. Custom data structures defined using Lisp's defstruct or object-oriented features could also be employed for more complex graphical entities, allowing for the encapsulation of data and associated drawing behaviors.

Lisp's functional nature and its support for recursion make it well-suited for defining drawing algorithms. For example, drawing a line using a simple algorithm like Bresenham's line algorithm could be implemented as a recursive function that iteratively determines which pixels to color based on the line's endpoints. Similarly, generating fractal geometries, which often exhibit self-similar patterns, can be elegantly achieved using recursive functions that define the iterative steps of the fractal generation process.28

Considering the minimal set of primitives required to build a more complex graphics system in Lisp, discussions on minimal Lisp implementations offer some insight.38 These discussions often highlight fundamental list operations (cons, car, cdr), conditional forms (if, cond), and basic functions as the core building blocks of the language. In the context of drawing, one could imagine implementing basic pixel manipulation using these primitives, perhaps by interacting with the underlying operating system's graphics APIs through Lisp's foreign function interface (FFI). From there, higher-level drawing functions for lines, shapes, and text could be built upon this foundation. The ability to manipulate lists could be fundamental for representing sequences of points that define lines or polygons.

## **5. Exploring Graphics Libraries in Common Lisp**

For developers seeking to create graphical applications in Common Lisp without delving into the intricacies of low-level implementations, a variety of robust open-source graphics libraries are available. These libraries provide pre-built functionalities, abstracting away the complexities of underlying graphics APIs and offering higher-level drawing primitives that simplify the development process.

One prominent library is **Sketch** 40, a Common Lisp framework inspired by the Processing language, geared towards electronic art, visual design, and game prototyping.42 Sketch leverages the SDL2 library as its rendering backend 42, providing a cross-platform foundation for its graphics capabilities. It offers a range of drawing primitives, including rect for rectangles, circle and ellipse for circular shapes, line and polyline for drawing lines, polygon for arbitrary polygons, and ngon for regular polygons.42

Another significant library is **CL-GD** 40, which provides an interface to the GD Graphics Library, a widely used library for the dynamic creation of images.44 CL-GD focuses primarily on image manipulation and supports output to various formats such as JPEG, PNG, and GIF.44 It offers functions for drawing basic geometric shapes like lines (draw-line), rectangles (draw-rectangle), circles (draw-filled-circle), polygons (draw-polygon), and arcs (draw-arc), as well as functions for drawing text (draw-string).44

**Vecto** 40 is a simple vector drawing library for Common Lisp that focuses on outputting PNG files.40 It provides a function-oriented interface reminiscent of the CL-PDF library.51 Vecto supports drawing vector paths using functions like move-to, line-to, curve-to, and arc, and allows for painting these paths using stroke and fill operations. It also includes functionality for drawing text using specified fonts.51

The **Lispbuilder-sdl** project 40 is an umbrella project that provides a comprehensive set of libraries for game and multimedia development in Common Lisp. Its core component, lispbuilder-sdl, offers bindings and Lispy abstractions for the SDL library.52 This library provides a wide array of 2D graphics primitives, including functions for drawing lines (draw-line), boxes (draw-box), circles (draw-circle), ellipses (draw-ellipse), polygons (draw-polygon), and Bezier and Catmull-Rom curves (draw-bezier, draw-curve).55 Furthermore, Lispbuilder-sdl includes separate packages like lispbuilder-sdl-image for loading various image formats 52 and lispbuilder-sdl-ttf for rendering TrueType fonts.52

Beyond these, other notable Common Lisp graphics libraries include **CLinch** 40, a 2D/3D graphics engine built on OpenGL, and **cl-cairo2** 40, which provides bindings to the Cairo graphics library, known for its high-quality 2D rendering capabilities. The availability of these diverse libraries underscores the active and multifaceted Lisp graphics community, catering to a wide spectrum of application requirements.

To provide a clearer overview, the following table summarizes the key features of the discussed Common Lisp graphics libraries:

| **Library Name** | **Underlying Technology** | **Output Formats** | **Basic Drawing Primitives** | **Text Support** | **Interactive Graphics** | **Notable Features** |
| --- | --- | --- | --- | --- | --- | --- |
| Sketch | SDL2 | Window display | rect, circle, ellipse, line, polygon | Yes | Yes | Inspired by Processing, focused on art and prototyping |
| CL-GD | GD Graphics Library | JPEG, PNG, GIF, etc. | line, rectangle, circle, polygon, arc | Yes | No | Image manipulation, dynamic image creation |
| Vecto | Native Lisp (CL-Vectors) | PNG | paths (lines, curves, arcs) | Yes | No | Simple vector drawing, PostScript/PDF-like interface |
| Lispbuilder-sdl | SDL | Window display, various image files | line, box, circle, ellipse, polygon, curves | Yes | Yes | Comprehensive for game and multimedia development, image and font loading |
| CLinch | OpenGL | Window display | 2D and 3D primitives | Yes | Yes | OpenGL-based 2D/3D graphics engine |
| cl-cairo2 | Cairo | Various image and surface formats | paths, shapes, text | Yes | Yes | High-quality 2D rendering, vector graphics |

## **6. Drawing with AutoLISP in AutoCAD**

AutoLISP stands as a specific and widely utilized dialect of Lisp designed for extending the functionality of AutoCAD, a leading commercial CAD software.29 Its integration into AutoCAD provides a powerful scripting language that enables users to automate tasks, customize the user interface, and create specialized drawing tools within the familiar AutoCAD environment. AutoLISP serves as a compelling example of Lisp's practical application in a professional graphics domain.

AutoLISP offers a range of primitive drawing commands that directly correspond to AutoCAD's native drawing entities. These commands, such as LINE, CIRCLE, RECTANG, ARC, and TEXT, are typically accessed and executed within AutoLISP programs using the COMMAND function.14 The COMMAND function takes the name of an AutoCAD command as its first argument, followed by the necessary parameters for that command. For instance, to draw a circle with its center at coordinates (0,0) and a radius of 5, one would use the AutoLISP expression: (command "\_.CIRCLE" "0,0" 5).14 The \_. prefix ensures that the standard AutoCAD command is invoked, regardless of any potential user-defined command with the same name.

AutoLISP allows for the creation of custom commands using the defun function.31 This enables users to define their own specialized drawing routines. For example, one could define a new command C:POINTLABEL that prompts the user to pick a point and then draws a marker at that point along with a text label displaying its coordinates.29 AutoLISP also provides functions for interacting with the AutoCAD drawing environment, such as getpoint, which allows the user to graphically select a point in the drawing.29 The value returned by getpoint can then be used as input for drawing commands.

Beyond simply executing AutoCAD commands, AutoLISP offers deeper integration with AutoCAD's object model through extensions like Visual LISP (VLISP).29 This allows programmers to directly access and manipulate AutoCAD's internal data structures representing graphical entities. Functions like vlax-get-acad-object and vla-get-documents enable AutoLISP programs to retrieve the active AutoCAD application object and the collection of open documents, respectively.30 This level of access allows for more sophisticated control over drawings, enabling tasks such as iterating through all circles in a drawing, modifying their properties (e.g., color, radius), or creating custom routines that react to specific events within the AutoCAD environment.

## **7. The Era of Lisp Machines and Advanced Graphics**

The 1980s witnessed the rise of Lisp machines, a unique class of computers specifically designed to execute the Lisp programming language with high efficiency.4 This era represented a distinctive approach to computing where hardware architectures were tailored to the demands of a high-level language, particularly addressing the memory-intensive nature of symbolic computation and graphics prevalent in Lisp environments.

Among the various vendors of Lisp machines, Symbolics stands out for its pioneering work in advanced graphics capabilities.11 Symbolics developed comprehensive hardware and software solutions for both 2D and 3D computer graphics.24 Their S-Graphics suite, a Lisp-programmable 3D graphics environment, achieved considerable success and was utilized in television and animation studios.23 These machines boasted advanced features for their time, including high-resolution bit-mapped displays, sophisticated windowing systems, and dedicated hardware support for graphics rendering.22

The Symbolics Genera operating system 64, written entirely in Lisp, provided a rich and integrated development environment for creating complex Lisp-based applications, including those in the realm of graphics. Genera featured a presentation-based window system known as Dynamic Windows 64, advanced object-oriented programming capabilities through Flavors and the Common Lisp Object System (CLOS) 65, and sophisticated debugging tools.64 These features collectively facilitated the development of intricate graphical software on Symbolics machines.

While Symbolics is perhaps the most recognized name in Lisp machine graphics, other vendors also contributed to this era. Lisp Machines Incorporated (LMI) and Xerox, with its Interlisp-D workstations 22, also offered Lisp-based systems that often included graphical capabilities. The existence of multiple vendors during this period underscores the significant interest and innovation surrounding Lisp-based computing and its applications in graphics.

## **8. Conclusion: Lisp's Enduring Relevance in Graphics**

This report has explored the multifaceted relationship between Lisp and computer graphics, highlighting Lisp's historical significance, its core features that lend themselves to graphical applications, and its continued relevance in the field. Despite not achieving the mainstream adoption in graphics seen by some other programming languages, Lisp offers a unique combination of power and flexibility that sustains its presence in specific niches.

Lisp's enduring relevance in computer graphics stems from its inherent power and versatility. Its flexible syntax, centered around S-expressions, and its powerful macro system enable the creation of highly expressive and domain-specific languages for graphics tasks. The functional programming paradigm encouraged by Lisp can lead to elegant and maintainable solutions for complex graphical algorithms. Furthermore, the availability of robust open-source graphics libraries for Common Lisp, such as Sketch, CL-GD, Vecto, and Lispbuilder-sdl, provides developers with the tools necessary to build a wide range of graphical applications without the need for low-level programming.

The historical impact of Lisp machines, particularly those from Symbolics, underscores Lisp's potential for driving advanced graphics technologies. These machines pioneered many of the graphical computing concepts we rely on today, demonstrating Lisp's capability to power sophisticated visual systems. Even the continued use of AutoLISP within AutoCAD highlights the practical value of Lisp for extending and automating tasks within a professional graphics environment.

Looking to the future, there appears to be a persistent and dedicated community interested in leveraging Lisp for graphics development. Ongoing projects, such as new open-source Common Lisp 3D graphics initiatives 73, and renewed discussions about developing advanced 3D systems in Lisp 23 suggest a continued, albeit perhaps niche, presence in the domain. Lisp's unique метапрограммирование capabilities and its historical legacy in interactive graphics position it as a language with the potential to inspire new approaches and innovative solutions in the ever-evolving field of computer graphics.

#### Works cited

1. Brief Study of LISP, accessed on May 12, 2025, <https://pnrsolution.org/Datacenter/Vol4/Issue3/63.pdf>
2. history - University of Bath, accessed on May 12, 2025, <http://www.bath.ac.uk/~masjap/TYL/history.html>
3. Lisp (programming language) - Wikipedia, accessed on May 12, 2025, <https://en.wikipedia.org/wiki/Lisp_(programming_language)>
4. CSc 520 Principles of Programming Languages History of the Lisp Language History of the Lisp Language. . . History of the Lisp L - Computer Science, accessed on May 12, 2025, <https://www.cs.arizona.edu/~collberg/Teaching/520/2005/Handouts/Handout-4.pdf>
5. History of Lisp - John McCarthy, accessed on May 12, 2025, <http://jmc.stanford.edu/articles/lisp.html>
6. How Lisp Became God's Own Programming Language - Two-Bit History, accessed on May 12, 2025, <https://twobithistory.org/2018/10/14/lisp.html>
7. History of Lisp - John McCarthy, accessed on May 12, 2025, <http://jmc.stanford.edu/articles/lisp/lisp.pdf>
8. Lisp: Background and History - Department of Computer Science and Engineering, accessed on May 12, 2025, <https://cse.buffalo.edu/~eblanton/course/cse410-2025-0s/materials/06-lisp-history.pdf>
9. The Evolution of Lisp - Dreamsongs, accessed on May 12, 2025, <https://dreamsongs.com/Files/Hopl2.pdf>
10. How the strengths of Lisp-family languages facilitate building complex and flexible bioinformatics applications - PMC, accessed on May 12, 2025, <https://pmc.ncbi.nlm.nih.gov/articles/PMC5952920/>
11. Lisp has a long history in computer graphics. It was particularly popular in the... | Hacker News, accessed on May 12, 2025, <https://news.ycombinator.com/item?id=23212974>
12. What was the historical significance of Lisp from a language design standpoint? - Quora, accessed on May 12, 2025, <https://www.quora.com/What-was-the-historical-significance-of-Lisp-from-a-language-design-standpoint>
13. Journal articles: 'LISP (Computer program language)' - Grafiati, accessed on May 12, 2025, <https://www.grafiati.com/en/literature-selections/lisp-computer-program-language/journal/>
14. AutoCAD 2025 Help | Tutorial: Getting Started (AutoLISP) | Autodesk, accessed on May 12, 2025, <https://help.autodesk.com/view/ACD/2025/ENU/?guid=GUID-C64046FA-CD9E-4B38-9967-A501119E4A62>
15. COMMON LISP: A Gentle Introduction to Symbolic Computation, accessed on May 12, 2025, <https://www.cs.cmu.edu/~dst/LispBook/book.pdf>
16. The Rise & Fall of LISP - Too Good For The Rest Of the World - Reddit, accessed on May 12, 2025, <https://www.reddit.com/r/lisp/comments/11ttnxk/the_rise_fall_of_lisp_too_good_for_the_rest_of/>
17. New LISPer recommended papers : r/lisp - Reddit, accessed on May 12, 2025, <https://www.reddit.com/r/lisp/comments/g35r66/new_lisper_recommended_papers/>
18. How Lisp Became God's Own Programming Language - Hacker News, accessed on May 12, 2025, <https://news.ycombinator.com/item?id=18225870>
19. A curated list of awesome Scheme libraries and resources - GitHub, accessed on May 12, 2025, <https://github.com/schemedoc/awesome-scheme>
20. (PDF) The future of Lisp. - ResearchGate, accessed on May 12, 2025, <https://www.researchgate.net/publication/220462702_The_future_of_Lisp>
21. The architecture of lisp machines (Journal Article) - OSTI.GOV, accessed on May 12, 2025, <https://www.osti.gov/biblio/6684161>
22. Lisp machine - Wikipedia, accessed on May 12, 2025, <https://en.wikipedia.org/wiki/Lisp_machine>
23. Symbolics Lisp Machines Graphics Demo (1990) [video] - Hacker News, accessed on May 12, 2025, <https://news.ycombinator.com/item?id=36016649>
24. Symbolics Lisp Machines Graphics Demo (1990) - Reddit, accessed on May 12, 2025, <https://www.reddit.com/r/lisp/comments/13o5r1l/symbolics_lisp_machines_graphics_demo_1990/>
25. The Evolution of Lisp - Dreamsongs, accessed on May 12, 2025, <https://www.dreamsongs.com/Files/HOPL2-Uncut.pdf>
26. What did Alan Kay mean by, 'Lisp is the greatest single programming language ever designed'? - Quora, accessed on May 12, 2025, <https://www.quora.com/What-did-Alan-Kay-mean-by-Lisp-is-the-greatest-single-programming-language-ever-designed>
27. Refurb weekend: the Symbolics MacIvory Lisp machine I have hated, accessed on May 12, 2025, <http://oldvcr.blogspot.com/2024/10/refurb-weekend-symbolics-macivory-lisp.html>
28. Journal articles: 'LISP (computer programming language)' - Grafiati, accessed on May 12, 2025, <https://www.grafiati.com/en/literature-selections/lisp-computer-programming-language/journal/>
29. AutoLISP - Wikipedia, accessed on May 12, 2025, <https://en.wikipedia.org/wiki/AutoLISP>
30. Chapters 1 and 2. Introduction to AutoLISP/Visual LISP, accessed on May 12, 2025, <http://lispexpert.blogspot.com/p/blog-page.html>
31. AutoCAD LT 2025 Help | Tutorial: Creating a New Custom Command and Controlling with System Variables (AutoLISP) | Autodesk, accessed on May 12, 2025, <https://help.autodesk.com/view/ACDLT/2025/ENU/?guid=GUID-1330DB99-DDFA-44D0-BDEB-676FF619EBE5>
32. AutoLISP Quick Start - AfraLISP, accessed on May 12, 2025, <https://www.afralisp.net/autolisp/tutorials/quick-start.php>
33. AutoLISP Basics - YouTube, accessed on May 12, 2025, <https://www.youtube.com/watch?v=SkHnD9xQTPE>
34. BEGINNER AUTOLISP - LESSON 1 - INTRODUCTION - YouTube, accessed on May 12, 2025, <https://www.youtube.com/watch?v=kofj18DbcXc&pp=0gcJCdgAo7VqN5tD>
35. AEDD260Lecture07 Introduction to AutoLISP programming - YouTube, accessed on May 12, 2025, <https://www.youtube.com/watch?v=mVW7KBy9Yk0>
36. Introducing AutoLISP for AutoCAD LT 2024 - YouTube, accessed on May 12, 2025, <https://www.youtube.com/watch?v=3KYKR0ILwZQ>
37. Lisp as a second language,composing programs and music., accessed on May 12, 2025, <http://recherche.ircam.fr/equipes/repmus/LispSecondLanguage/OO2/OO2.html>
38. What are the minimal set of primitives a Lisp implementation can be built on, one complete enough even though it may be slow? - Reddit, accessed on May 12, 2025, <https://www.reddit.com/r/lisp/comments/18zpzlw/what_are_the_minimal_set_of_primitives_a_lisp/>
39. How much Lisp do you have to implement as primitives before you can implement the rest of Lisp in Lisp - Reddit, accessed on May 12, 2025, <https://www.reddit.com/r/lisp/comments/srnp4t/how_much_lisp_do_you_have_to_implement_as/>
40. Common Lisp libraries overview, accessed on May 12, 2025, <https://common-lisp.net/libraries>
41. basic graphics library : r/Common\_Lisp - Reddit, accessed on May 12, 2025, <https://www.reddit.com/r/Common_Lisp/comments/ylzxwu/basic_graphics_library/>
42. vydd/sketch: A Common Lisp framework for the creation of electronic art, visual design, game prototyping, game making, computer graphics, exploration of human-computer interaction, and more. - GitHub, accessed on May 12, 2025, <https://github.com/vydd/sketch>
43. luksamuk/game-sketch: Common Lisp system for creating sketches, inspired by the Processing 2.x API. - GitHub, accessed on May 12, 2025, <https://github.com/luksamuk/game-sketch>
44. CL-GD - Use the GD Graphics library from Common Lisp, accessed on May 12, 2025, <https://edicl.github.io/cl-gd/>
45. Common Lisp Libraries by Edi Weitz | A community project to maintain and improve libraries developed originally by Edi Weitz, accessed on May 12, 2025, <https://edicl.github.io/>
46. CL-GD - Use the GD Graphics library from Common Lisp - Quicklisp, accessed on May 12, 2025, <https://www.quicklisp.org/beta/UNOFFICIAL/docs/cl-gd/doc/index.html>
47. SKIPPY - Read and write GIF files with Common Lisp, accessed on May 12, 2025, <https://www.xach.com/lisp/skippy/>
48. vindarel/curated-awesome-cl: A curated and non exhaustive list of Common Lisp libraries and resources. - GitHub, accessed on May 12, 2025, <https://github.com/vindarel/curated-awesome-cl>
49. CodyReichert/awesome-cl: A curated list of awesome Common Lisp frameworks, libraries and other shiny stuff. - GitHub, accessed on May 12, 2025, <https://github.com/CodyReichert/awesome-cl>
50. How to create a simple graphic using Common Lisp? - Stack Overflow, accessed on May 12, 2025, <https://stackoverflow.com/questions/75203851/how-to-create-a-simple-graphic-using-common-lisp>
51. Vecto - Simple Vector Drawing with Common Lisp - Zach Beane, accessed on May 12, 2025, <https://www.xach.com/lisp/vecto/>
52. Lispbuilder provides a range of libraries for developing useful portable Common Lisp applications - GitHub, accessed on May 12, 2025, <https://github.com/lispbuilder/lispbuilder>
53. Video game graphics in Common Lisp - narkive, accessed on May 12, 2025, <https://comp.lang.lisp.narkive.com/VHl7wB7A/video-game-graphics-in-common-lisp>
54. LISPBUILDER-SDL-IMAGE -, accessed on May 12, 2025, <https://lispbuilder.sourceforge.net/lispbuilder-sdl-image.html>
55. LISPBUILDER-SDL - a cool library - Quicklisp, accessed on May 12, 2025, <https://www.quicklisp.org/beta/UNOFFICIAL/docs/lispbuilder/doc/lispbuilder-sdl.html>
56. LISPBUILDER-SDL-GFX - - Quicklisp, accessed on May 12, 2025, <https://www.quicklisp.org/beta/UNOFFICIAL/docs/lispbuilder/doc/lispbuilder-sdl-gfx.html>
57. LISPBUILDER-SDL-TTF -, accessed on May 12, 2025, <https://lispbuilder.github.io/documentation/lispbuilder-sdl-ttf.html>
58. lispbuilder-sdl - CLiki, accessed on May 12, 2025, <https://www.cliki.net/lispbuilder-sdl>
59. SDL2 Tutorial: How To Draw and Render Using C++ - YouTube, accessed on May 12, 2025, <https://www.youtube.com/watch?v=mxWkj0KiICk>
60. Graphics library - CLiki, accessed on May 12, 2025, <https://www.cliki.net/graphics%20library>
61. Batch-processing AutoCAD drawings from LISP without SDI - Through the Interface, accessed on May 12, 2025, <https://www.keanw.com/2009/06/batch-processing-autocad-drawings-from-lisp-without-sdi.html>
62. Solved: Open drawing by lisp - Autodesk Community, accessed on May 12, 2025, <https://forums.autodesk.com/t5/visual-lisp-autolisp-and-general/open-drawing-by-lisp/td-p/13023055>
63. Symbolics Architecture - People @EECS, accessed on May 12, 2025, <http://people.eecs.berkeley.edu/~kubitron/courses/cs252-F00/handouts/papers/symbolics.pdf>
64. Symbolics Lisp Machine Museum, accessed on May 12, 2025, <https://www.chai.uni-hamburg.de/~moeller/symbolics-info/>
65. Symbolics - Wikipedia, accessed on May 12, 2025, <https://en.wikipedia.org/wiki/Symbolics>
66. Symbolics Technical Summary, accessed on May 12, 2025, <https://smbx.org/symbolics-technical-summary/>
67. Genera (operating system) - Wikipedia, accessed on May 12, 2025, <https://en.wikipedia.org/wiki/Genera_(operating_system)>
68. Genera User's Guide Overview of Symbolics Computers Documentation Notation Conventions Modifier Key Conventions, accessed on May 12, 2025, <https://bitsavers.trailing-edge.com/pdf/symbolics/software/genera_8/Genera_User_s_Guide.pdf>
69. Programming the User Interface Preface to Programming the User Interface top-level macros, cp:define-command and dw:define-progr - Bitsavers.org, accessed on May 12, 2025, <https://bitsavers.org/pdf/symbolics/software/genera_8/Programming_the_User_Interface.pdf>
70. Symbolics X Window System User's Guide The Genera X Client Configuring the Remote Screen Facility for the X Client Installing - Bitsavers.org, accessed on May 12, 2025, <https://bitsavers.org/pdf/symbolics/software/genera_8/Symbolics_X_Window_System_User_s_Guide.pdf>
71. Ask HN: What Were the Differences Between Symbolics Genera and Xerox Interlisp-D, accessed on May 12, 2025, <https://news.ycombinator.com/item?id=36713595>
72. What was so good about Genera? I read a piratebay comment about how it was the best IDE ever. Obviously, this was hyperbole, but there has to be some truth to it. : r/lisp - Reddit, accessed on May 12, 2025, <https://www.reddit.com/r/lisp/comments/fvzai/what_was_so_good_about_genera_i_read_a_piratebay/>
73. New open source Common Lisp 3D graphics project -- call for participation - Reddit, accessed on May 12, 2025, <https://www.reddit.com/r/lisp/comments/weocc5/new_open_source_common_lisp_3d_graphics_project/>