SLAC-PUB-358
October 1967

# A Picture Calculus*

W. F. Miller and Alan C. Shaw
Stanford University, Stanford, California



(For presentation at the conference on "Emerging Concepts in Computer Graphics", University of Illinois, Urbana, Illinois, November 5-8, 1967.)

* This work supported in part by the National Science Foundation (Grant No. GP-7615) and in part by the U. S. Atomic Energy Commission.

# I. INTRODUCTION

This work represents what we think are some substantial steps toward a picture calculus.[1] This picture calculus brings together under a common formalism the treatment of picture recognition and picture generation for a broad class of pictures. It is comprised of a picture description language (PDL),[2] rules for transforming and comparing structures represented in PDL, data structures and control for generation of pictures, and the parsing and primitive recognizers needed for picture recognition. The types of pictures that most obviously lend themselves to treatment with this calculus are what we call artificial pictures as opposed to natural pictures. Line drawings, flow charts, block letters, printed pages, particle physics pictures, and closed boundary curves can be meaningfully described in PDL.[3]

Two general programs have been developed to date that employ elements of this picture calculus. The first has been developed for the recognition of particle physics pictures and other pictures having a simple graph structure; the second has been developed to permit drawing and transformation of line drawings on a CRT.

# II. THE PICTURE DESCRIPTION LANGUAGE (PDL)

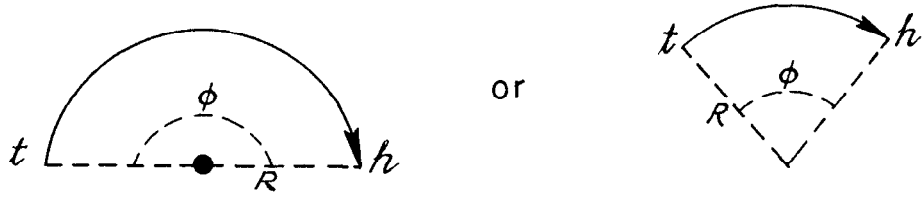
## A. The Primitives

The picture description language PDL permits one to describe the concatenation of the primitive elements of a picture. The choice of primitives will depend on the particular application. A primitive may be defined as any two- (n-) dimensional object with two distinguished points, a tail and a head. In general, a primitive will be a picture that can be handled more conveniently as a unit than in terms of its subparts. Primitives are concatenated together only at their heads and/or tails. Abstractly, a picture described in PDL can be represented as a labeled, directed graph with the primitives as directed edges pointing from tail to head. We often refer to the graph of a picture.

Picture primitives are specified in terms of primitive classes which in turn are specified by an attribute list. The attribute list contains the class name, a tail and head specification, and an arbitrary list of additional

attributes. As an example we may define the primitive class of arcs of circles of any radius, negative curvature, and arc less than 180°. The attribute of the primitive class ARC has the form

ARC ≡ (ARC, Counterclockwise limit, Clockwise limit, Curvature < 0, φ < 180°)




This is a specific instance of the general form

PRIMITIVE CLASS ≡ (<NAME>, <tail specification>, <head specification>, <1st attribute>, <2nd attribute>, ... <nth attribute>)

A superscript label identifies a particular member of a primitive class. An element $ARC^{\dagger}$ of the class ARC, for example, has a value list containing specific values for each attribute on the attribute list of the class. E.g.,

$$VALUE(ARC^{\dagger}) = (ARC^{\dagger}, \vec{X}_{TAIL}, \vec{X}_{HEAD}, (CURVATURE =)-2, (\varphi =)60°)$$

There may be redundant information on the attribute list. Some of the attributes may be irrelevant for some uses. However, for generality of form and application all information is retained.

We allow blank (invisible) and don't care primitives. They may be used for connecting disjoint parts of a picture or to specify the geometrical relationship between parts of a picture. One special primitive, the null point primitive, λ, plays a special role. It consists only of a tail and head with identical position and it is represented as a labeled node in a graph.

B. The Syntax

A sentence, S, in the language is defined by

1. $S \rightarrow p \mid (S\ \theta\ S) \mid (\sim S) \mid (\overline{S}) \mid T(\omega)S \mid S^{\ell}$
2. $\theta \rightarrow + \mid \times \mid - \mid * \mid \sim$
3. p is a primitive class described in A above.
4. $\{+, \times, -, *\}$ are concatenation operators described in C below.
5. $\{\sim, \overline{\ }, T(\omega)\}$ are unary operators described in D and E below.
6. $\ell$ is a label designator illustrated in G below.

C. The Concatenation Operators

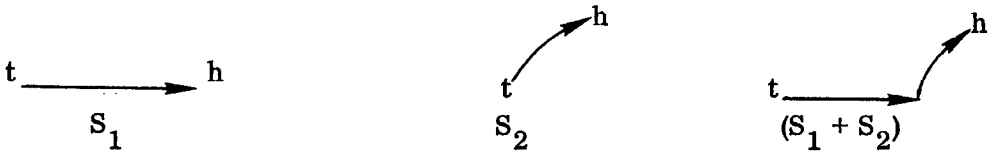
The concatenation operators +, ×, -, *, and ~ are binary operators defined below. In all cases

$$\begin{cases} \text{Tail}\ ((S_1\ \theta\ S_2)) = \text{Tail}\ (S_1) \\ \text{Head}\ ((S_1\ \theta\ S_2)) = \text{Head}\ (S_2), \quad \theta \in \{+, \times, -, *, \sim\}\ . \end{cases}$$

7. The + operator: head to tail:

$(S_1 + S_2)$ concatenates the head of $S_1$ to the tail of $S_2$.

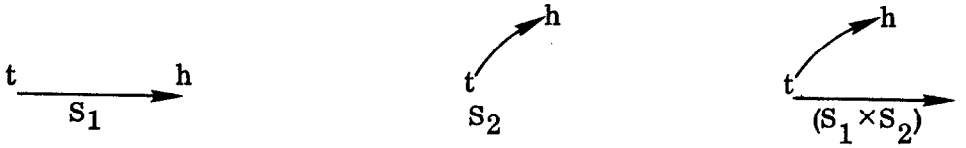
Example:




8. The × operator: tail to tail:

$(S_1 \times S_2)$ concatenates the tail of $S_1$ to the tail of $S_2$.

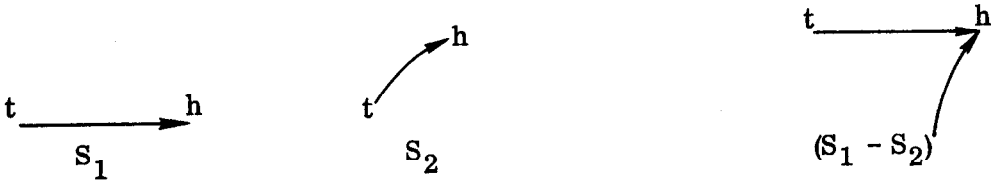
Example:




9. The - operator: head to head:

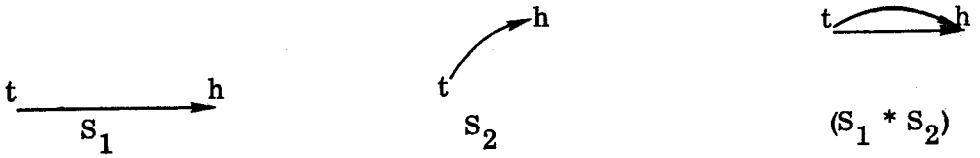
$(S_1 - S_2)$ concatenates the head of $S_1$ to the head of $S_2$.

Example:




10. The * operator: head to head and tail to tail:

$(S_1 * S_2)$ concatenates the tail of $S_1$ to the tail of $S_2$ and the head of $S_1$ to the head of $S_2$.




The * operation may be undefined for some combinations of structures, S, just as the arithmetic operation a/b is undefined for certain values of a or b.

Example:

VECT1 ≡ (VECT1, tail at origin, head at upper right, unit vector in first quadrant)

VECT2 ≡ (VECT2, tail at origin, head at lower left, unit vector in third quadrant)

(VECT1 * VECT2) is undefined.

11. The binary ~ operator:

$(S_1 \sim S_2) \equiv (S_1 + (\sim S_2))$ where $(\sim S_2)$ is defined in D.

That is, the binary ~ means simply + the unary ~ of $S_2$, where the unary ~ is as defined in D.

D. The Unary Operators

The operators ~ and ⌐ are unary operators defined as follows:

12. The unary ~ operator: switches head and tail.

Tail ((~ S))= Head (S)

Head ((~ S))= Tail (S)

The structure remains the same.

Example:


S (~S)

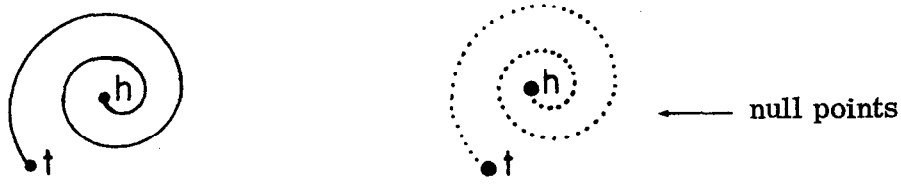
13. The ⌐ operator: blanks out all points.

Head ($\ulcorner S$) = Head (S)

Tail ($\ulcorner S$) = Tail (S)

All points in the structure are turned to null points.

Example:




S ($\ulcorner S$)

E. The Transformation Operator $T(\omega)$ is a unary operator which may operate on a particular primitive $p_j^\dagger$, or on a class of primitives $p_j$. $T(\omega)$ represents the affine transformations.

14. $\vec{X}' = M\vec{X} + \vec{T}$

where $\vec{X}$ is any point in the structure, $\vec{X}'$ is the corresponding point in the transformed structure, M is a matrix of constants, and $\vec{T}$ is a constant vector. This includes stretching, rigid body rotations and translations, and shearing transformations.

Although it may not be very useful, we define a transformation on a class.

$$T(\omega)\ (p) \equiv \left\{ T(\omega)\ p^\dagger \mid p^\dagger\ \epsilon p \right\}$$

A common $T(\omega)$ will be isotropic stretching, i.e., scalar multiplication. We shall indicate that particular $T(\omega)$ by:

$Cp_j$; C is a positive constant.

We observe that

15. $T(\omega)\ (S^\dagger\ (p_i^\dagger,\ p_j^\dagger\ \ldots)) = S^\dagger\ (T(\omega)p_i^\dagger,\ T(\omega)p_j^\dagger,\ \ldots)$

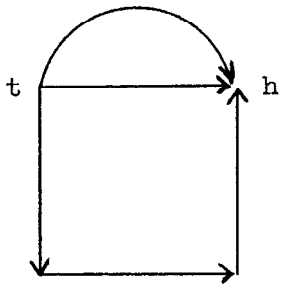
F. Uses and Illustrations

Before proceeding with further description of the language, let us examine some uses and illustrations.

We can describe a class of houses with any of two types of roofs permitted. We need the following primitive classes which have a single member as shown.


In the generation of a display we may wish to show the house with the round roof. That is

$$H_2 \rightarrow ((p_6 * p_3) * ((p_4 + p_3) + p_5))$$

We would show 


A class of houses H is described by the following sentences in our language.

16. $H \rightarrow (R * B)$
$R \rightarrow R1 | R2$
$R1 \rightarrow ((p_1 + p_2) * p_3)$
$R2 \rightarrow (p_6 * p_3)$
$B \rightarrow ((p_4 + p_3) + p_5)$

A pattern recognizer driven by the above grammar will find either a house with a round roof or a house with a triangular roof according to whichever is present. Shaw[3] calls the process of searching a picture according to an explicit grammar "picture parsing".

A practical picture parser consists of a library of primitive recognizer subroutines and a general PDL parser. Input to the program consists of digital data representing the picture, and a PDL grammar representing the picture classes to be found. The output consists of the VALUE's of the primitives found, the string (PDL) description of the picture, and the parsing tree.

We may observe that the picture description language is in a sense a metalanguage. A particular set of sentences in PDL constitutes a smaller language. In writing down a description of a class of pictures to be recognized, one is writing a grammar for this class of pictures. The picture parser must decide whether the object picture is in the class to be recognized. That is, it must recognize whether this picture has a structure in the grammar. If the answer to the picture parse is "TRUE", the parser can exhibit the particular sentence found and its tree.

Let us illustrate with a simple particle physics picture. We consider a class of interactions starting with a negative particle which may pass

through the picture or may scatter from a positive particle or may decay into a neutral and another negative particle. Thus

$$17.\quad T_- \rightarrow t_- \,|(t_- + (T_- \times T_+))|(t_- + (T_- \times T_n))$$

where $T_-$ is a negative track with all subsequent events, $t_-$ is a primitive negative track, $T_+$ is a positive track with all subsequent interactions, and $T_n$ is a neutral track (not seen) with all subsequent interactions.

Let us consider only positive particles that continue through the chamber, that is,

$$18.\quad T_+ \rightarrow t_+$$

where $t_+$ is a primitive positive track.

Let us consider neutral particles that may decay into pairs, that is,

$$19.\quad T_n \rightarrow t_n \,|(t_n + (T_+ \times T_-)).$$

A picture that must start with a negative track would be represented by

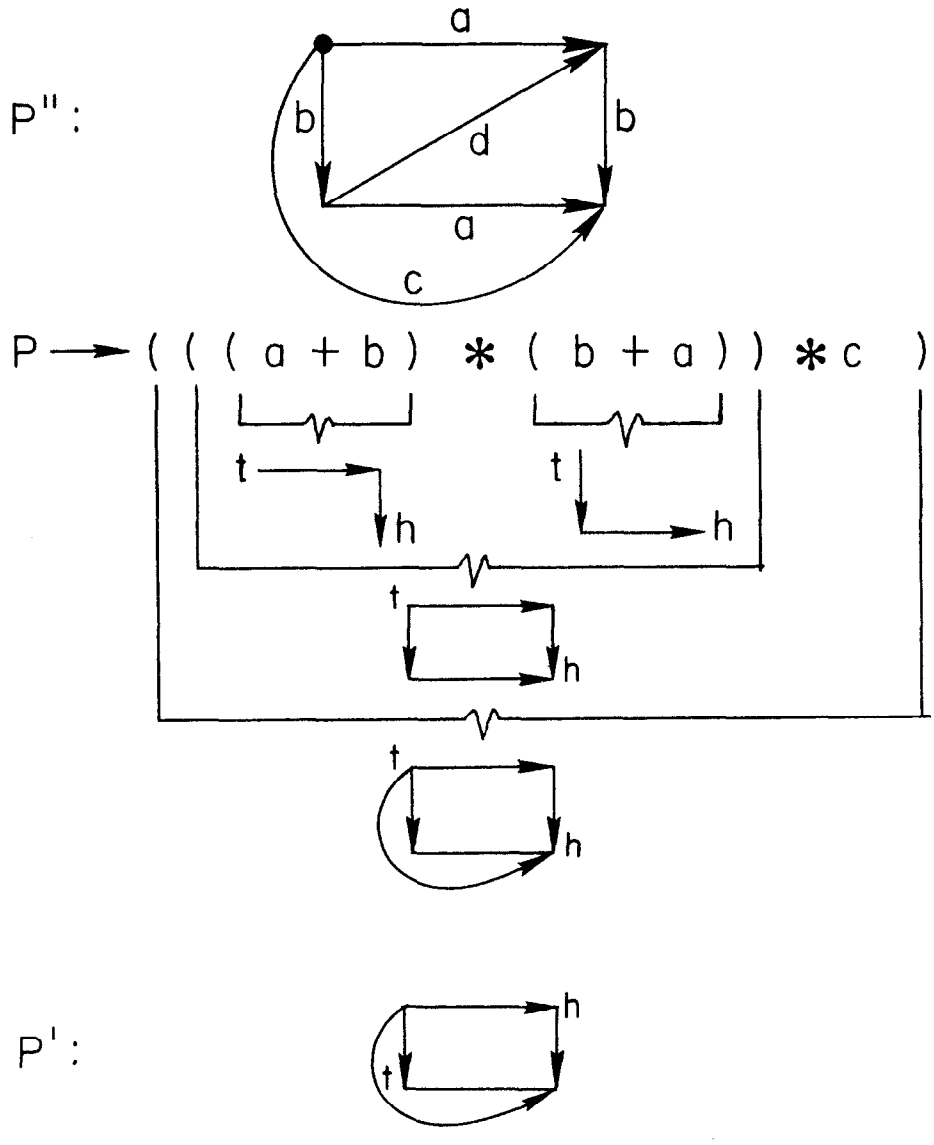
$$20.\quad \begin{aligned} P &\rightarrow T_- \\ T_- &\rightarrow t_- \,|(t_- + (T_- \times T_+))|(t_- + (T_- \times T_n)) \\ T_+ &\rightarrow t_+ \\ T_n &\rightarrow t_n \,|(t_n + (T_+ \times T_-)) \end{aligned}$$

Figure 1 shows a picture that would be generated by sentences in the grammar (20.).

The picture description language set forth up to here is suitable for describing pictures with a simple connectivity. It can not yet describe all connected graphs. We introduce a labeling scheme which, with the unary operations ~ and $\ulcorner$ , permit the description of any connected graph.

## G. Labeling

The labeling scheme presented here is edge-oriented and preserves the identity of the edge through the various transformations and manipulations which operators and operands may undergo. Consider the example in Ref. 2.




$P' \rightarrow (((\overline{(\sim b^{\alpha})}) + P) + (\overline{(\sim b^{\beta})}))$

where $b^{\alpha}$ and $b^{\beta}$ refer to the left and right b's respectively in the picture above of P''. We go back in P and label the b's in order that the head and tail of P' will be where we want them.

$P \rightarrow (((a + b^{\beta}) * (b^{\alpha} + a)) * c)$

$P'' \rightarrow (P' * d)$

The labeling preserves the correct identification even though we may perform various allowable operations such as commutation.

Shaw[4] has defined equivalence of structures represented in PDL in terms of the graph of a picture.

Weak Equivalence:

$S_1$ is weakly equivalent to $S_2$ ($S_1 \equiv_w S_2$) if there exists an isomorphism between the graphs of $S_1$ and $S_2$ such that the corresponding edges have identical names.

Equivalence:

$S_1$ is equivalent to $S_2$ ($S_1 \equiv S_2$) if

(a) $S_1 \equiv_w S_2$ and

(b) $\text{Tail}(S_1) = \text{Tail}(S_2)$

(c) $\text{Head}(S_1) = \text{Head}(S_2)$

## III. OPERATOR RULES

In carrying out substitutions and transformations of structures represented in PDL the following algebraic properties of the operators are useful.

### A. Associativity

The binary operators are associative.

(1) $((S_1 + S_2) + S_3) \equiv (S_1 + (S_2 + S_3))$

(2) $((S_1 \times S_2) \times S_3) \equiv (S_1 \times (S_2 \times S_3))$

(3) $((S_1 - S_2) - S_3) \equiv (S_1 - (S_2 - S_3))$

(4) $((S_1 * S_2) * S_3) \equiv (S_1 * (S_2 * S_3))$

### B. Commutativity

(1) * is the only commutative operator

$(S_1 * S_2) \equiv (S_2 * S_1)$

(2) $\times$ and - are weakly commutative, that is,

$(S_1 \times S_2) \equiv_w (S_2 \times S_1)$

$(S_1 - S_2) \equiv_w (S_2 - S_1)$

C. The ~ Operator

~ acts like complementation in a Boolean Algebra:

(1) $(\sim (S_1 + S_2)) \equiv ((\sim S_2) + (\sim S_1))$
$(\sim (S_1 * S_2)) \equiv ((\sim S_2) * (\sim S_1))$

(2) ~ obeys a "de Morgan's law" with respect to $\times$ and - :

$(\sim (S_1 \times S_2)) \equiv ((\sim S_2) - (\sim S_1))$
$(\sim (S_1 - S_2)) \equiv ((\sim S_2) \times (\sim S_1))$

D. The Blanking Operator $\ulcorner$

(1) $(\overline{\ulcorner (\ulcorner S)}) \equiv (\ulcorner S)$

(2) $(\overline{\ulcorner (S_1 \ \theta \ S_2)}) \equiv ((\overline{\ulcorner S_1}) \ \theta \ (\overline{\ulcorner S_2}))$
$\theta \epsilon \{+, \times, -, *\}$

(3) $(\overline{\ulcorner (\sim S)}) \equiv (\sim (\ulcorner S))$

E. The Null Point Primitive

(1) $(S \ \theta \ \lambda) \equiv (\lambda \ \theta \ S)$ $\quad \theta \epsilon \{+, \times, -, *\}$

(2) $(S \ \varphi \ \lambda) \equiv S$ $\quad \varphi \epsilon \{+, \times, -\}$

(3) $(\sim \lambda) \equiv \lambda$

(4) $(\lambda \ \theta \ \lambda) \equiv \lambda$

## IV. PICTURE RECOGNITION

Shaw[3] describes a general picture parsing algorithm for pictures described in PDL. He also describes the implementation of a parser for a subset of PDL called SPDL which was suitable for experimenting with particle physics pictures.

One of the reasons for developing a formal grammar for pictures is that one can then utilize the powerful methods already developed for the analysis of string languages. The particular nature of picture processing suggests a goal-oriented or top-down analysis scheme is preferable to a bottom-up scheme.[3]

In analogy to language parsing, the "terminal" symbols of the grammar are the picture primitives. A picture parse is directed toward recognizing those primitives which are grouped together in allowable structures according to the grammar. A top-down parse of a picture will commence with a starting symbol P and generate from left to right sentences S of the language according to the grammar. When the goal is a primitive class name, a search routine is called which in turn calls a pattern recognizer to determine whether a primitive of the class sought is located at the specified location. If there is a primitive present its value is returned to the value list.

For example, in parsing the grammar (16.) of a house one comes to the parsing of

$$R1 \rightarrow ((p_1 + p_2) * p_3)$$

When the terminal symbol $p_1$ is the goal, the $p_1$ recognizer is called. When $p_2$ is the goal, the search routine directs the $p_2$ recognizer to determine whether there is a $p_2$ primitive with a tail at the head of the $p_1^{\dagger}$ already found, since $p_2$ is concatenated to $p_1$ by the + operator. Similarly, when $p_3$ is the goal, the search routine directs the $p_3$ recognizer to look for a $p_3$ primitive with tail at the tail of $(p_1 + p_2)^{\dagger}$ and head at the head of $(p_1 + p_2)^{\dagger}$ since $p_3$ is concatenated to $(p_1 + p_2)$ by the * operator.

An extensive discussion of the advantages and disadvantages of a top-down parse is given in Reference 3.

The picture parsing method has been applied to a small number of spark chamber photographs. Figure 2 shows a typical photograph from a 600 MeV electron-electron scattering experiment after digitization by the Hummingbird film digitizer.[5] Electron beams were supplied by the Stanford Mark III accelerator. They were circulated in opposite directions in two storage rings and collided together in a common section. The scattering is observed via a set of spark chambers and counters. Each scattered electron traverses successively a 6-gap spark chamber, a 4-gap spark chamber, and a shower chamber. The possible points of interaction lie along the horizontal line in the center of the figure. Figure 3 contains the result of the picture parse. The display shows an abstracted version of what has been recognized.

Each primitive (including blank ones) is represented by a straight line segment terminating on the tail and head coordinates of the primitive. The "T" and "H" indicate the tail and head of the last primitive found.

## V. PICTURE GENERATION

The picture calculus rules have been utilized as the basis of two picture generation programs.[6,7] The first of these was done as an exercise to gain experience with an interactive program. The second one has the goal of providing a facility for making drawings interactively. The program permits one to write PDL expressions from the console and it will display the represented pictures.

The picture description is stored as a string. One can then parse the string description according to the general PDL syntax each time one makes a change in the picture and generate a vector representation in the picture buffer. Concatenation of new structures to existing ones is done by adding to the string description. Replacement of single primitives can now be done by changing the primitive definition. Insertion and deletion of more complex structures has not yet been implemented. The data structure holds both a string description and an explicit vector description for substructures. As long as one continues to deal with this structure as a whole only the vector description is of interest.

## VI. CONCLUSIONS

It would appear that a picture description language of the PDL type is capable of describing a large number of different classes of artificial pictures. The development of formal operator rules combined with the labeling scheme permits one to manipulate the string description into convenient forms and the $T(\omega)$ operator permits one to transform pictures.

We intend to explore further the uses of PDL for picture recognition and to expand the calculating and transforming capability of our drawing programs. We have done very little with the $T(\omega)$ operator except translation. We are developing more fully the $T(\omega)$ rules. We are also trying to develop a more general description language that will permit (or require) conditional concatenations.

Implementation of the recognition programs has been in terms of FORTRAN subroutines. Implementation of the drawing programs has been in terms of PL/1 procedures. We expect to unify these capabilities into a programming language for various aspects of picture processing.

References

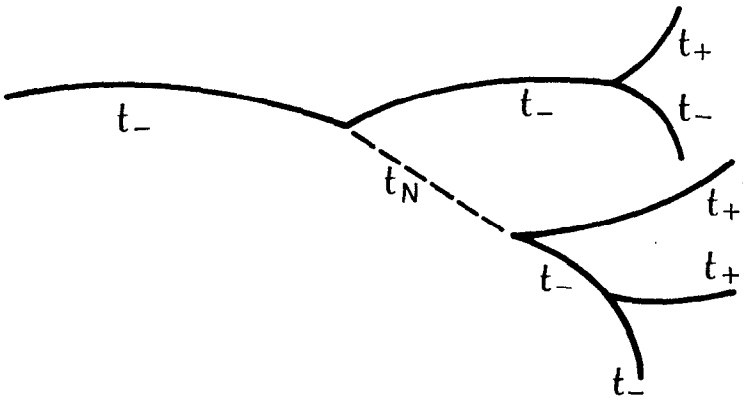

1. W. F. Miller and Alan C. Shaw, "A Picture Calculus", GSG Memo No. 40, Computation Group, Stanford Linear Accelerator Center, Stanford University, June 1967.
2. Alan C. Shaw, "A Proposed Language for the Formal Description of Pictures", GSG Memo No. 28, Computation Group, Stanford Linear Accelerator Center, Stanford University, February 1967.
3. Alan C. Shaw, "The Formal Description and Parsing of Pictures", Ph.D. Thesis, Stanford University (in press).
4. Alan C. Shaw, "A Picture Calculus - Further Definitions and Some Basic Theorems", GSG Memo No. 46, Computation Group, Stanford Linear Accelerator Center, Stanford University, June 1967.
5. J. van der Lans, "Hummingbird, Automatic Film Digitizers at the Stanford Linear Accelerator Center", Proceedings of the 1967 International Conference on Programming for Flying Spot Devices, Powell, B. W. and Seyboth, P. (Ed.), January 1967, 51-61. Also published as SLAC-PUB-251, Stanford Linear Accelerator Center.
6. Y. Noyelle, "Implementation on the PDP-1 of a Subset of the Picture Calculus", Term Project CS250, Computer Science Department, Stanford University, June 1967.
7. J. George, "Picture Generation Based on the Picture Calculus", GSG Memo No. 50, Computation Group, Stanford Linear Accelerator Center, Stanford University, October 1967.


Related Papers


1. R. Narasimhan, "A Linguistic Approach to Pattern Recognition", Report No. 121, Digital Computer Laboratory, University of Illinois, Urbana, Illinois, July 1962.
2. R. A. Narasimhan, "Syntax-Directed Interpretation of Classes of Pictures", Comm. ACM 9, p. 166-173, March 1966.
3. R. S. Ledley, "Programming and Utilizing Digital Computers", McGraw-Hill, New York, 1963, Chapter 8.

Related Papers (contd)


4. R. S. Ledley, "High-Speed Automatic Analysis of Biomedical Pictures", Science 146, p. 216-223, October 1964.
5. R. A. Kirsch, "Computer Interpretation of English Text and Picture Patterns", IEEE Trans. Elec. Comp. EC-13, p. 363-376, August 1964.
6. Jerome Feder, "Linguistic Specification and Analysis of Classes of Patterns", Technical Report 400-147, New York University School of Engineering and Science, October 1966.
7. K.J. Breeding, "Pattern Grammar for a Pattern Description Language", Report No. 177, Department of Computer Science, University of Illinois, Urbana, Illinois, May 1965.
8. Robert H. Anderson, "Syntax-Directed Recognition of Hand-Printed Two-Dimensional Mathematics", Presented at ACM Symposium on Interactive Systems for Experimental Applied Mathematics, August 1967.
9. M. B. Clowes, "A Generative Picture Grammar", Seminar Paper No. 6, Computing Research Section, Commonwealth Scientific and Industrial Research Organization, Canberra City, Australia, April 1966.




$$P \longrightarrow \left(t_- + \left(\left(t_- + \left(t_+ \times t_-\right)\right) \times \left(t_N + \left(t_+ \times \left(t_- + \left(t_+ \times t_-\right)\right)\right)\right)\right)\right)$$

867A4

FIG. 1 -- PARTICLE PHYSICS PICTURE

FIG. 2--Colliding beam experiment frame after digitization.

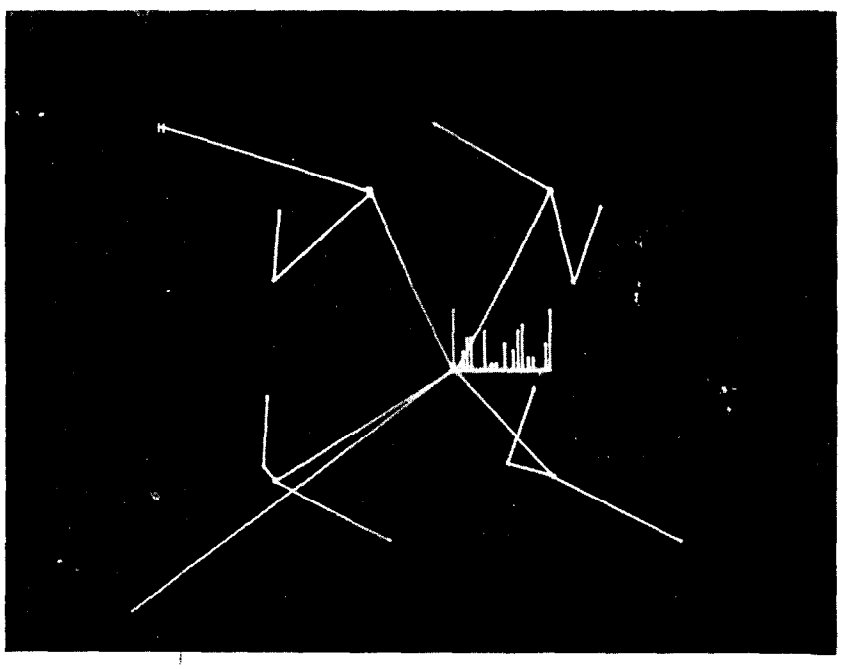


FIG. 3--Result of parsing picture in Fig. 2.